Visualforce Basics

Monday, August 08, 2016 10:27 AM

Get Stated with Visualforce

Vforce defined:

a web-dev framework that enables developers to build custom user interfaces for mobile and desktop apps that can be hosted on the Force.com platform.

You can display a VF page within a standard page layout, a tab, a Salesforce1 layout, a custom button or link, or a standard button or link. More often than not, we link the Visualforce page via a unique URL.

Create and Edit Vforce pages

Vforce markup can be mixed with HTML markup, CSS styles and JavaScript libraries. Creating Vforce pages in Dev. Console allows you to use automatic syntax highlighting, tag pair matching, auto-suggest and auto-complete.

Vforce page blocks allows you to add components within the page.
Markup:

```
<apex: pageBlock>
    <apex: pageBlockSection>
        <apex: pageBlockSectionItem>
        </>
    </…Section>
</…Block>
```

Use Simple Variables and Formulas

Vforce allows you to display dynamic data on the page depending on certain criteria. These are called visualforce expressions. You can use any set of literal values, variables, sub-expressions or operators within these expressions. You cannot use a method call.

Visualforce expression syntax: {! *Expression* }

Whitespace is ignored inside the delimiter. The **resulting value** can be a primitive, a Boolean, an sObject, a controller method or other items.

Global Variables - expression syntax: {! *$GlobalName.fieldName* }. VF expressions are case-insensitive.
All global variables you're allowed to reference: https://developer.salesforce.com/docs/atlas.en-us.206.0.pages.meta/pages/pages_variables_global.htm

Formula Expressions - When you need to manipulate a result, use a formula to represent it correctly.

Use operators to join expressions together to create compound expressions.

Operators must be used within Visualforce expression syntax to be evaluated.Visualforce supports the following operators.

Math Operators

Operator	Description	Use
+	Calculates the sum of two values.	value1 + value2 and replace each value with merge fields, expressions, or other numeric values.
-	Calculates the difference of two values.	value1 - value2 and replace each value with merge fields, expressions, or other numeric values.
*	Multiplies its values.	value1 * value2 and replace each value with merge fields, expressions, or other numeric values.
/	Divides its values.	value1 / value2 and replace each value with merge fields, expressions, or other numeric values.
^	Raises a number to a power of a specified number.	number^integer and replace number with a merge field, expression, or another numeric value; replace integer with a merge field that contains an integer, expression, or any integer.
()	Specifies that the expressions within the open parenthesis and close parenthesis are evaluated first. All other expressions are evaluated using standard operator precedence.	(expression1) expression2... and replace each expression with merge fields, expressions, or other numeric values.

Logical Operators

You can't have a relative comparison expression that includes a null value. Doing so results in an exception. Specifically, you can't have a null value on either side of the following operators:

- < (less than)
- <= (less than or equals)
- > (greater than)
- >= (greater than or equals)

Operator	Description	Use
= and ==	Evaluates if two values are equivalent. The = and == operator are interchangeable.	expression1=expression2 or expression1 == expression2, and replace each expression with merge fields, expressions, or other numeric values.
<> and !=	Evaluates if two values are not equivalent.	expression1 <> expression2 or expression1 != expression2, and replace each expression with merge fields, expressions, or other numeric values.
<	Evaluates if a value is less than the value that follows this symbol.	value1 < value2 and replace each value with merge fields, expressions, or other numeric values.
>	Evaluates if a value is greater than the value that	value1 > value2 and replace each value with merge

	follows this symbol.	fields, expressions, or other numeric values.
<=	Evaluates if a value is less than or equal to the value that follows this symbol.	value1 <= value2 and replace each value with merge fields, expressions, or other numeric values.
>=	Evaluates if a value is greater than or equal to the value that follows this symbol.	value1 >= value2 and replace each value with merge fields, expressions, or other numeric values.
&&	Evaluates if two values or expressions are both true. Use this operator as an alternative to the logical function AND.	(logical1) && (logical2) and replace logical1 and logical2 with the values or expressions that you want evaluated.
||	Evaluates if at least one of multiple values or expressions is true. Use this operator as an alternative to the logical function OR.	(logical1) || (logical2) and replace any number of logical references with the values or expressions you want evaluated.

Text Operators

Operator	Description	Use
&	Connects two or more strings.	string1&string2 and replace each string with merge fields, expressions, or other values.

Function Syntax: https://developer.salesforce.com/docs/atlas.en-us.206.0.pages.meta/pages/pages_variables_functions.htm

Use Standard Controllers

Model - the database
View - Visual Force page
Controller - code in-between

Add the following attribute to display data from a record:
StandardController='StandardField'
The standard controller includes dml actions such as create, edit, save, and delete.

When traversing across custom related records, use the API name, not the __r notation.
Twitter bootstrap and Salesforce: https://developer.salesforce.com/blogs/developer-relations/2014/03/twitter-bootstrap-and-visualforce-in-minutes.html

Display Records, Fields, and Tables

Output components are components that output data from a record and enable you to design a view-only user interface.
In order to display record details, add the following markup:
<apex:detail />. This is assuming the standard controller is set.
You can adjust the attributes in the markup.

Use the <apex:relatedList> markup to show lists of records related to the data. You can even set the page size and other attributes inside the markup.

Use <apex:outputField> markup to add fields individually. To format this data better, surround the output fields within an <apex:pageBlockSection> and an <apex:pageBlock>

Use <apex:pageBlockTable> markup to add a table of data to a page. <apex:dataList> and <apex:dataTable> are iteration components for creating tables and lists without the platform styling. <apex:repeat> is an iteration component that you use to generate any arbitrary markup for a collection of records.

<apex:enhancedList> and <apex:listViews> can be used instead of <apex:relatedList>

Input Data Using Forms

Use <apex:form> to create the form framework. Use <apex:inputField> to create a page to edit data. Use <apex:commandButton> to create a button with an action tied to a standard controller.
<apex:form> packages everything inside it (similar to Ajax) and send the data back to Salesforce.
<apex:pageMessages/> adds a message to the location you suggest it to go.

Use Standard List Controllers

Standard List Controllers allow you to create Visualforce pages that can display or act on a set of records. You can query records into this list without altering back-end code. Filtering and pagination through the results are also actions you can take.

Use <apex:pageBlockTable> to display a list of records. Set the var attribute to the variable name you want to use to access the record's values.

Use {! listViewOptions } to get a list of list view filters available for an object. Use {! filterId } to set the list view filter to use for a standard list controller's results.

Solution for Trailhead Challenge:

```
<apex:page standardController="Account" recordSetVar="accounts">
  <apex:pageBlock title="Account Links" id="account_links">
        <apex:repeat value="{! accounts}" var="a">
                        <li>
           <apex:outputLink value="/{!a.id}">
                  {!a.Name}
                  </apex:outputLink>
                  </li>
      <br />
              </apex:repeat>
  </apex:pageBlock>
</apex:page>
```

Use Static Resources

Static resources allow you to upload content (image, javascript library, etc.) inside a Visualforce page. Caching and distribution are handled automatically.

You can upload .js and even .zip! Files.

Using Static resources from .zip file:

```
<apex:stylesheet value="{!URLFOR($Resource.jQueryMobile, 'jquery.mobile-1.4.5.css') }"/>
   <apex:includeScript value="{! $Resource.jQuery }"/>
   <apex:includeScript value="{!URLFOR($Resource.jQueryMobile, 'jquery.mobile-1.4.5.js') }"/>
```

This is similar to adding components to a <head> markup.

Solution for Trailhead Challenge:

```
<apex:page >
        <apex:image alt="imageTest" title="imageTest" value="{! URLFOR($Resource.vfimagetest, 'cats/kitten1.jpg') }"/>
</apex:page>
```

Create and Use Custom Controllers

Custom Controllers allow you to fully customize the logic and data manipulation that can be used by a VF page. You can make external web service callouts using custom controllers.

Custom controllers don't inherit from another class or implement an interface promising to conform to the requirements of a VF controller.

Instead of the plain text, try this markup: <apex:outputText value="{! $ObjectType.Contact.Fields.FirstName.Label }"/>

Getter methods pull data out of your controller onto your page. There are corresponding setter methods that let you submit values from the page back up to your controller. Like getter methods, you prefix your setters with “set”, and other than that, they’re just methods that take an argument.

An alternative to getters and setters is to use Apex properties. Properties are kind of a combination of a variable with getter and setter methods, with a syntax that groups them together more clearly. A simple property that references a custom object might be declared like this.