



P r o f e s s i o n a l E x p e r t i s e D i s t i l l e d

Visualforce Developer's Guide

Learn the latest developments in Salesforce with this hands-on pocket guide

W.A.Chamil Madusanka

[PACKT] enterprise 
PUBLISHING professional expertise distilled

www.allitebooks.com

Visualforce Developer's Guide

Learn the latest developments in Salesforce with this hands-on pocket guide

W.A.Chamil Madusanka



BIRMINGHAM - MUMBAI

Visualforce Developer's Guide

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: October 2013

Production Reference: 1161013

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78217-981-8

www.packtpub.com

Cover Image by Aashish Variava (aashishvariava@hotmail.com)

Credits

Author

W.A.Chamil Madusanka

Project Coordinator

Akash Poojary

Reviewers

Aruna Lambat

Rahul Sharma

Niket Soral

Proofreaders

Dirk Manuel

Stephen Copestake

Acquisition Editor

Rubal Kaur

Indexer

Tejal Soni

Commissioning Editors

Shreerang Deshpande

Shaon Basu

Production Coordinator

Shantanu Zagade

Technical Editor

Krishnaveni Haridas

Cover Work

Shantanu Zagade

About the Author

W.A.Chamil Madusanka is a Salesforce.com certified Force.com developer. He has been working on Force.com projects since 2011. He is working as a developer for many custom applications built on Force.com and has also trained end users and new Salesforce developers in his current company (Attune Lanka (pvt) Ltd.) and former company (Sabre Technologies (pvt) Ltd.). He has won the Salesforce New Year Resolution 2013 challenge which was rolled out by Salesforce. He is an active member of the Force.com community and he has been contributing to the Force.com community through various channels. He is avid about Force.com and shares his knowledge on Force.com technologies through his blog (<http://salesforceworld.blogspot.com/>). He is a super-contributor on the Force.com discussion board and shares his knowledge and experience on Force.com by providing effective solutions to developer questions. He is the initiator and the group leader of the Sri Lanka Salesforce Platform Developer User Group. His contribution to the Sri Lanka Salesforce community has led to an increase in Salesforce competency in Sri Lanka. He completed his B.Sc in Computer Science from the University of Colombo School of Computing, Sri Lanka (UCSC). His areas of interest include Cloud computing, semantic web technologies, and Ontology-based systems. Hailing from Polonnaruwa, which is an ancient city in Sri Lanka, he currently resides in Gampaha which is located in the Western province of Sri Lanka. His interests include reading technology books and technology blog posts, and playing cricket. Chamil can be reached via Twitter (@chamilmadusanka), Skype (chamilmadusanka), and e-mail (chamil.madusanka@gmail.com).

Acknowledgments

I would like to express my gratitude to many people who saw me through this book; to all those who provided support, read and offered their comments, allowed me to quote their remarks, and proofread. I would like to thank Chandima Cooray, Sriyangi Perera, and Asitha Siriwardhana who introduced me to the Salesforce path. Their support and guidance has been a great strength when I started to work on Salesforce technologies. I want to thank my family, who supported and encouraged me and accepted the moments when I was away from them. I would like to thank Packt Publishing for giving me the opportunity to write my first book. I would like to thank Daniel D'Abreo, Ameya Sawant, Siddhant Shetty, Shreerang Deshpande, Akash Poojary, Shaon Basu, and Krishnaveni Nair from Packt Publishing for helping me throughout the process of completing the book. I would like to thank Aruna Lambat, Rahul Sharma, and Niket Soral for their technical review of this book and Udaya K. Jayawardhana, Priyanke Wijesekara, and Pushpani Nawarathna for proofreading this book and for their valuable comments.

About the Reviewers

Aruna Lambat is a profound Technical Lead working on Salesforce.com technology with an insightful understanding of software design and development. She is passionate about building better products and providing excellent services leading to healthier customer satisfaction. She started working on the Salesforce.com platform in 2008. She entered into IT acquaintance in 2004 as a student. She completed her Master's degree in Computer Applications from Maharashtra, India. She is associated with the IT industry since 2007, having started her carrier as a Java developer and later shifted her focus to Cloud computing specifically in Salesforce.com. She has been a Salesforce Certified Developer (DEV401), Administrator (ADM201) and Advanced Administrator (ADM301/211), providing regular contributions to the Salesforce developer community. She is also certified for her knowledge in Java as a **Sun Certificated Java developer (SCJP)** and **Web component developer (SCWCD)**. Before contributing to this book as a reviewer, she worked previously as a technical reviewer for two Salesforce books: *Visualforce Development Cookbook* and *Force.com Tips and Tricks*. She helped the author by citing an example in the *Force.com Developer Certification Handbook (DEV401)* book. Aruna works with a reputed India-based IT group MNC, which is primarily engaged in providing a range of outsourcing services, business process outsourcing, and infrastructure services. Aruna works as a Lead Consultant / Salesforce Application Architect in the Salesforce.com technology-based customer services. Aruna resides in Pune, the cultural capital of Maharashtra also known for its educational facilities and relative prosperity. She is from Nagpur, the Orange city, and her parents stay in the heart of Orange City. She completed her education in this city and achieved success at different points in her carrier with immense support from her parents. Aruna loves travelling (nature visits), reading fiction books, playing Pool, and roaming with friends during her free time.

My special thanks to my parents Mr. and Mrs. Anandrao Lambat, for always being there with me, for their immense help and support, and for guiding me through each and every step making it so enlightened.

Rahul Sharma has been working on Force.com projects since 2009. He is working as a developer, analyst, and consultant for many custom applications built on Force.com. He is an active member of the Force.com community and has completed his Bachelor of Engineering. He resides in Mumbai (India). He participates in various online coding challenges for learning new platforms and technologies.

Niket Soral is a Salesforce consultant from India having more than four years of experience. In addition to Salesforce, he has good technical skills in Microsoft Dynamics, data migration, integration through web services, and so on. He has also developed some AppExchange products. He has done BSc (Bachelor of Science) and MCA (Master of Computer Applications). He writes technical blogs and contributes to community sites in his spare time. He is fond of listening to music and social networking.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Instant Updates on New Packt Books

Get notified! Find out when new books are published by following [@PacktEnterprise](https://twitter.com/PacktEnterprise) on Twitter, or the *Packt Enterprise* Facebook page.

Table of Contents

Preface	1
Chapter 1: Getting Started with Visualforce	7
The MVC model	8
Understanding Visualforce	8
The Visualforce architecture	9
Advantages of Visualforce	11
Visualforce development tools	13
Summary	13
Chapter 2: Controllers and Extensions	15
Standard controllers	16
How to use a standard controller with a Visualforce page	16
Standard controller actions	17
Standard list controllers	20
How to use a standard list controller with Visualforce	20
Standard list controller actions	21
Custom controllers and controller extensions	23
Understanding custom controllers	23
Building a custom controller	23
Understanding controller extension	26
Building a controller extension	27
Controller methods	28
Getter methods	29
Setter methods	29
Action methods	31
Working with large sets of data on the Visualforce page	31
Order of execution of a Visualforce page	32
Order of execution for a Visualforce page's get requests	32
Order of execution for a Visualforce page's postback requests	33

Validation rules and standard controllers/custom controllers	34
Using the transient keyword	35
Considerations for creating custom controllers and controller extensions	36
Summary	36
Chapter 3: Visualforce and Standard Web Development Technologies	37
Styling Visualforce pages	38
Salesforce styles	38
Custom styles	38
Using JavaScript in Visualforce pages	42
Accessing Visualforce components in JavaScript	42
JavaScript remoting for Apex controllers	44
Using jQuery in Visualforce pages	47
HTML5 and Visualforce pages	49
Summary	50
Chapter 4: Visualforce Custom Components	51
Understanding Visualforce custom components	51
Creating and using a custom component	52
Custom attributes and custom controllers	54
Summary	56
Chapter 5: Dynamic Visualforce Bindings	57
Using dynamic references with standard objects and custom objects	57
Referencing Apex Maps and Lists	61
Working with field sets	64
Summary	64
Chapter 6: Visualforce Charting	65
Limitations and considerations of Visualforce charting	66
How does Visualforce charting work	66
Providing chart data	68
Using the controller method	69
Using a JavaScript function	69
Using a JavaScript array	70
A complex chart with Visualforce charting	71
Summary	74

Chapter 7: Visualforce for Mobile	75
Understanding Salesforce Mobile	75
Salesforce Mobile and Visualforce Mobile supporting devices	76
Capabilities and limitations of the mobile application	77
Using Visualforce Mobile	78
Developing and mobilizing Visualforce pages	78
Best practices for building Visualforce Mobile pages for iPhone and BlackBerry	78
iPhone considerations	80
BlackBerry considerations	80
Developing cross-platform compatible pages	81
Using the JavaScript library	83
Building a mobile-ready Visualforce tab	85
Creating the mobile configuration	85
Summary	86
Chapter 8: Best Practices for Visualforce Developments	87
Accessing component IDs	88
Page block components	88
Controllers and controller extensions	89
Improving Visualforce's performance	89
Static resources	92
Rendering PDFs	92
Using component facets	93
Summary	94
Appendix: Security Tips for Apex and Visualforce Development	95
Security scanning tools	95
Force.com Security Source Scanner	96
Cross-site scripting (XSS)	97
Cross-site request forgery (CSRF)	98
SOQL injection	98
Data access control	100
Summary	100
Index	101

Preface

Visualforce Developer's Guide is a practical, hands-on pocket guide that provides a clear and simple guidance to develop Visualforce pages for the Force.com platform and for mobile applications. It also contains a single, continuous, real-world example with code samples. This book explains the Visualforce concepts and technical aspects in a simple manner.

Visualforce Developer's Guide covers the main topics, starting with the development of Visualforce for the Force.com platform, and continuing on to developing Visualforce pages for Mobile applications quickly and painlessly.

The Force.com platform can automatically generate user interfaces (standard pages), but in some cases you might need to build a more customized UI. Visualforce allows developers to build sophisticated and customized user interfaces that can be hosted natively on the Force.com platform. Visualforce is a framework that includes a tag-based markup language similar to HTML.

What this book covers

Chapter 1, Getting Started with Visualforce, explains the MVC model and the Visualforce architecture. We will be introduced to Visualforce pages. We will understand MVC architecture and discuss about Visualforce pages. Further we will understand the architecture of Visualforce pages. We will define the advantages of Visualforce pages and will get an idea about Visualforce development tools.

Chapter 2, Controllers and Extensions, introduces the types of controllers and extensions which can be used for Visualforce pages. We will understand the controller types, and see some examples of these.

Chapter 3, Visualforce and Standard Web Development Technologies, explains how to develop Visualforce pages with standard web development technologies such as CSS, JavaScript, jQuery, and HTML5. The usage of static resource for CSS, JavaScript, and jQuery are also included in the chapter.

Chapter 4, Visualforce Custom Components, gives an overview of Visualforce custom components and further explains how to create a custom component and the usage of custom components.

Chapter 5, Dynamic Visualforce Bindings, explains the usage of dynamic references with standard objects and custom objects, how to reference Apex Maps and Lists, and how to work with field sets.

Chapter 6, Visualforce Charting, discusses Visualforce charting, which is a collection of components that provides a simple and intuitive way to create charts in your Visualforce pages and custom components.

Chapter 7, Visualforce for Mobile, covers how to extend applications built on the Force.com platform to mobile devices by using Visualforce Mobile.

Chapter 8, Best Practices for Visualforce Developments, explains the best practices for Visualforce and Apex developments. We will look at how to improve the user experience, and examine some coding standards for Visualforce development.

Appendix, Security Tips for Apex and Visualforce Developments, provides some security tips for Visualforce and Apex developments. We will look at some tools to scan our code for security and quality and will learn some security vulnerabilities.

What you need for this book

The prerequisites for the Force.com platform are as follows:

- Basic knowledge of Internet/websites
- A free Developer Force account (if you don't have one, please visit: <https://events.developerforce.com/signup>)
- Basic knowledge of Visualforce

Who this book is for

Visualforce Developer's guide is not a complete reference or a bible for Visualforce development. This is a time-saving pocket guide for Visualforce which includes the most needed and used technical aspects of Visualforce developments. Therefore, this book is suitable for the Force.com developers who have a basic knowledge of Visualforce.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.


Code words in text are shown as follows: "We can reference a static resource by name in page markup by using the `$Resource` global variable instead of hardcoding document ID."


A block of code is set as follows:

```
<apex:page controller="TransientExampleController">
  Non Transient Date: {!t1} <br/>
  Transient Date      : {!t2} <br/>
  <apex:form >
    <apex:commandLink value="Refresh"/>
  </apex:form>
</apex:page>
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this:

"The **Component Reference** is the best place to explore different components and their uses".

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Started with Visualforce

Cloud computing has made significant changes to the IT/software development industry. Cloud platforms are one of the important directions of cloud computing. Cloud platforms allow the developers to develop apps and run them on the Cloud, including platforms for building on-demand applications and **platforms as services (PaaS)**. Salesforce.com has introduced the first on-demand platform called Force.com.

This chapter will introduce you to Visualforce. We will go through the MVC architecture and Visualforce. Furthermore, we will look at the architecture of Visualforce pages. We will define the advantages of Visualforce pages and will get an idea about Visualforce development tools.

This chapter covers the following topics:

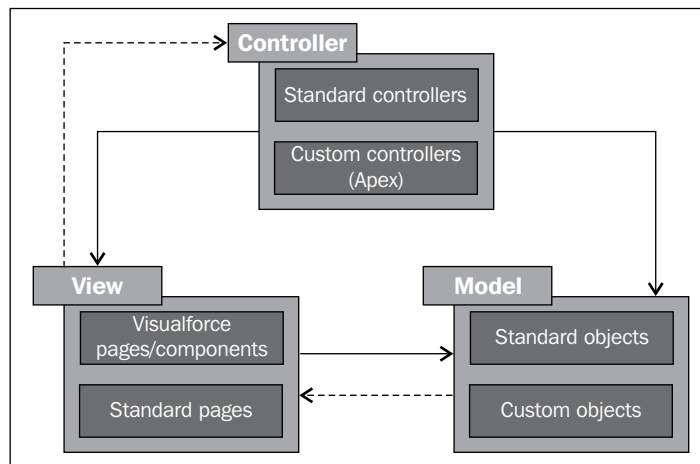
- The MVC model
- Understanding Visualforce
- Visualforce architecture
- Advantages of Visualforce
- Visualforce development tools

So let's get started and step into Visualforce.

The MVC model

The Force.com platform uses the **Model View Controller (MVC)** architectural pattern for developing an application. MVC splits the application development tools as follows:

- **Model:** This defines the structure of the data. In Force.com, objects define the data model. Salesforce has designed the platform by mapping every entity to some object.
- **View:** This defines how the data is represented. In Force.com, page layouts and Visualforce pages come under this category.
- **Controller:** This defines the business logic. The rules and actions which manipulate the data controls the view. In Force.com, **Apex** classes, triggers, workflows, approvals, and validation rules are under this category.



The MVC Architecture

Understanding Visualforce

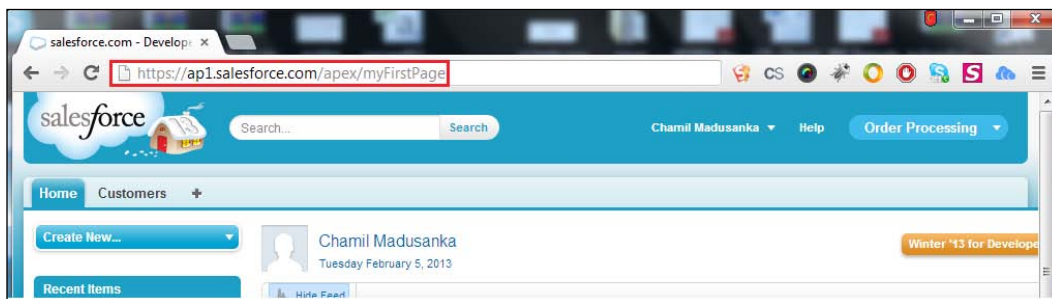
In the Force.com platform, we can develop Force.com applications with custom objects and standard objects. Every object has a standard user interface with one or more page layouts. But we cannot use standard page layouts for sophisticated requirements. Here, Visualforce comes into play.

Visualforce is a web-based UI framework, which can be used for building sophisticated, attractive, and dynamic custom user interfaces. Visualforce allows the developer to use standard web development technologies such as jQuery, JavaScript, CSS, and HTML5. Therefore, we can build rich UIs for any app including mobile apps. We'll be discussing about Visualforce with standard web development technologies and Visualforce for mobile in more depth later. Similar to HTML, the Visualforce framework includes a tag-based markup language.

A Visualforce page has two major elements called Visualforce markup and Visualforce controller. Visualforce markup consists of Visualforce tags with the prefix `apex:`, and there can be HTML tags, JavaScript, or any other standard web development code. Visualforce controller consists of a set of instructions to manipulate data and schema with the user interaction. It controls the interface as well. A standard controller, which is created along with the object can be used as Visualforce controller. A standard controller has the same logic and functionality which is used in standard pages. But when we need to use a different logic or functionality, we can write our own Apex controller class, and we can also write extensions to standard controllers or custom controllers using the Apex language. AJAX components, expression language formula for actions, and component binding interactions are there in Visualforce.

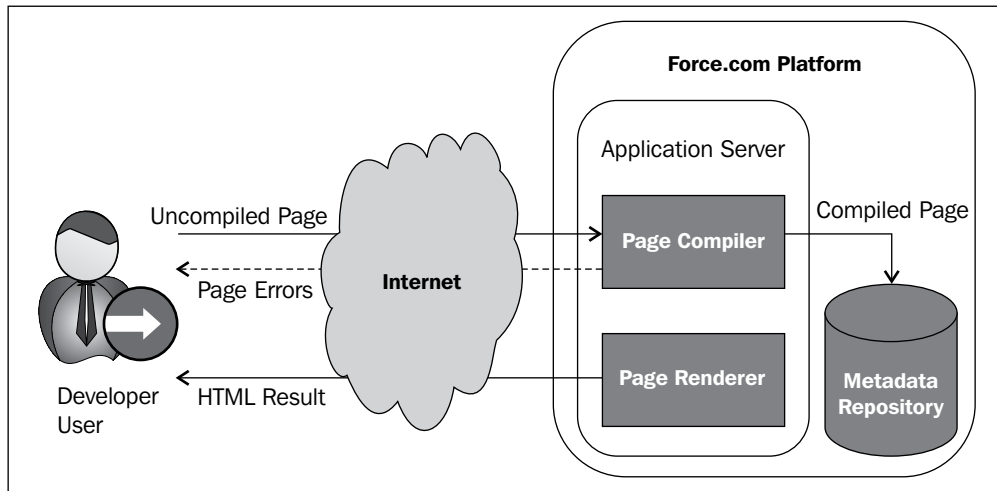
The Visualforce architecture

Visualforce is a markup language similar to HTML. Visualforce pages run on the Force.com platform and it can integrate with standard web development technologies such as JavaScript, jQuery, and styling by CSS. It allows us to build more rich and animated UIs. Each page can be identified by a unique page name. Therefore a Visualforce page has a unique URL for accessing the page. A developer can create a Visualforce page by entering the page name in the address bar, as shown in the following screenshot:



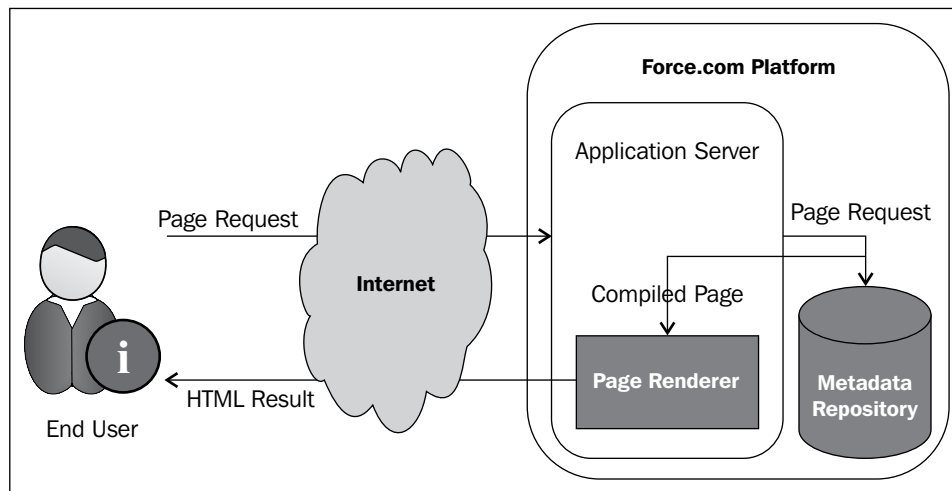
Create a Visualforce page from a URL

After creating the page, we can access the page by using the same URL. The Visualforce markup has a special set of components which is similar to the tag library system of other markups. These components allow us to create complicated components with a single tag. These components are processed and rendered on servers and finally delivered to the client. This methodology has higher performance and enriched functionality when compared to the client-only methods. A Visualforce page runs on the platform shown in following diagram:



Execution flow while saving Visualforce on a server

The preceding diagram illustrates that every time a developer saves a Visualforce page on a platform, the platform compiles the markup and related controllers. On successful compilation, markup is converted into an abstract set of instructions that can be understood by the Visualforce renderer. If there are any compilation errors, then it stops saving the page and returns the errors to the developer. If the saving attempt successfully finished, then the instructions are saved to the metadata repository and sent to the Visualforce renderer. The renderer converts the instructions into HTML, which can be understood by the browsers, and then refreshes the page as shown in the following diagram:



Execution flow of Visualforce page

The preceding diagram illustrates that when a non-developer user requests a Visualforce page, the application server retrieves the page from the metadata repository and then sends to the Visualforce renderer for HTML conversion. There is no compilation because the page has already been compiled into instructions during its development.

Advantages of Visualforce

The following are the advantages of Visualforce for a developer:

- **Model-View-Controller development style:** Visualforce adheres to the MVC pattern by providing the View of the application in the Force.com platform. A View is defined by user interfaces and Visualforce markup. The Visualforce controller which can be associated with Visualforce markup takes care of the business logic. Therefore, the designer and the developer can work separately, while the designer focuses on user interface and the developer focuses on business logic.
- **User-friendly development:** A developer (with an administrator profile) user can have a Visualforce editor pane at the bottom of every Visualforce page. This editor pane is controlled by the **Development Mode** option of the user record. This feature allows us to edit and see the resulting page at the same time and in the same window. This Visualforce editor has the code-saving feature with auto compilation and syntax highlighting.

- **A broad set of ready-to-serve Visualforce components:** Visualforce has a set of standard components in several categories. There are output components, for example, `<apex:outputPanel>`, `<apex:outputField>`, `<apex:outputText>`, `<apex:pageBlock>`, and so on. There are input components, for example, `<apex:inputFile>`, `<apex:inputField>`, `<apex:inputText>`, `<apex:selectList>`, and so on. These input and output components have a feature called data-driven defaults. For an example, when we specify the `<apex:inputField>` component in a particular Visualforce page, the `<apex:inputField>` tag provides the edit interface for that field with data-type-related widgets (for example, the **Date** field has the calendar, and the e-mail/phone fields have their particular validations). There are also AJAX components, for example, `<apex:actionStatus>`. AJAX components allow the user to enhance the level of interactivity for a particular interface.
- **Tightly integrated with Salesforce / Extends with custom components:** A Visualforce page can have a custom controller as well as a standard controller. A standard controller is created while creating the object and can be used for the Visualforce controller. A standard controller has the same logic and functionality which is used in standard pages. Visualforce pages adhere to these standardized methods and functionality. And we can also extend the standard components with custom components. For example, we can use an extension class for extending the standard controller of a particular Visualforce page. We can create our own Visualforce custom components instead of Visualforce in-built components for example, `<apex:inputFile>`, `<apex:inputField>`, `<apex:outputField>`, and so on. In the next two chapters we will discuss more about Visualforce controllers and Visualforce custom components.
- **Flexible and customizable with web technologies:** The Visualforce markup is more flexible and more customizable through the use of web technologies, for example, JavaScript, CSS, jQuery, Flash, and so on because it is eventually rendered into HTML. A designer can use the Visualforce tags with these web technologies.

Visualforce development tools

We can edit and view Visualforce pages from the set-up area by navigating to **Your Name | Setup | Develop | Pages**.

But, that's not the best way to develop Visualforce pages. There are a few other ways to build Visualforce pages, which are as follows:

- **The Visualforce editor pane:** This is discussed under the advantages of Visualforce.
- **The Force.com IDE:** This IDE is used for creating and editing Visualforce pages, custom Visualforce components, static resources, and controllers.
- **The Eclipse plugin for Force.com:** This is same as the Force.com IDE, we can use it to create and edit Visualforce pages, custom Visualforce components, static resources, and controllers are some of the major features of the Force.com IDE.

Summary

In this chapter we became familiar with the MVC model and Visualforce. We saw the Visualforce architecture and the advantages of Visualforce, and also the various Visualforce development tools.

2

Controllers and Extensions

A set of instructions that can react on the user's interaction with Visualforce markup (for example, a button click or a link click) is called as a controller. A controller can control the behavior of a page and it can be used to access the data which should be displayed on the page.

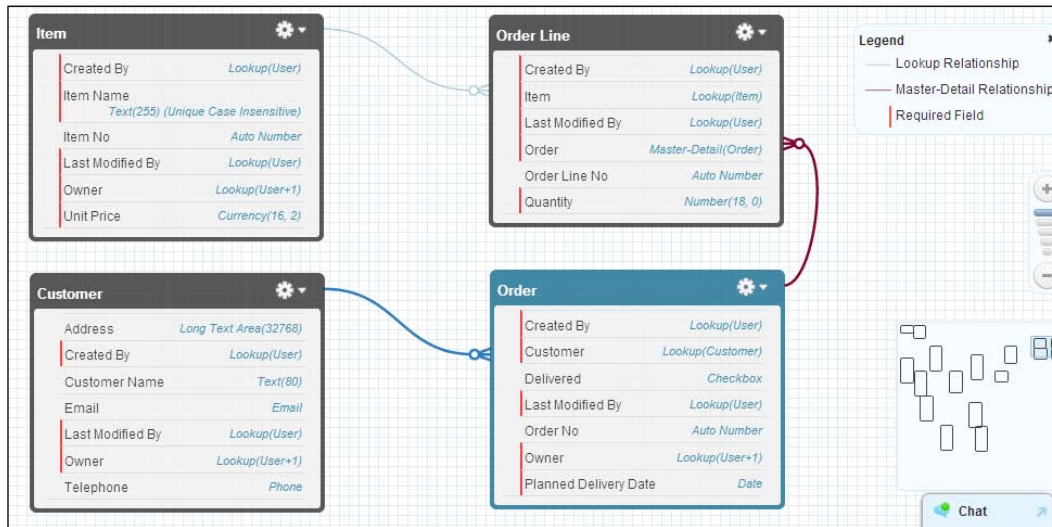
This chapter will introduce you to a few types of controllers and extensions that can be used for Visualforce pages. We will learn the types of controllers with examples.

This chapter covers the following topics:

- Standard controllers
- Standard list controllers
- Custom controllers and controller extensions
- Working with large sets of data on a Visualforce page
- Order of execution of a Visualforce page
- Validation rules and standard controllers or custom controllers
- Using the transient keyword
- Considerations for creating custom controllers and controller extensions

Let's look closer at controllers and extensions...

This chapter includes a set of examples to explain the important elements and features of Visualforce. Starting from this chapter we will build an order processing application. There are four custom objects (API names: `Customer__c`, `Item__c`, `Order__c`, `Order_Line__c`) in this application. The following is the E-R diagram of an order processing application which we will create on the Force.com platform:



The E-R diagram of an order Line processing application

Standard controllers


The Force.com platform provides a few types of controllers. The first one is standard controller and every sObject has a standard controller. They have the same logic and functionality as they are originally used in standard pages. Therefore we can use standard controllers with Visualforce pages. For example, if we use Contact standard controller for a Visualforce page, we can implement the standard `save` method for Contact without writing any additional Apex code. This behavior is the same as implementing the `save` method on the standard Contact edit page.

How to use a standard controller with a Visualforce page

The `<apex:page>` tag has an attribute called `standardController` which is used to associate a standard controller with a Visualforce Page. The value of the `standardController` attribute would be the API name of an sObject:

```
<apex:page standardController="Customer__c">
</apex:page>
```

The preceding code shows the usage of the `standardController` attribute.


 You cannot use the `standardController` and `controller` attributes at the same time.

Standard controller actions

In Visualforce pages, we can define the `action` attribute for the following standard Visualforce components:

- `<apex:commandButton>`: This component creates a button that calls an action
- `<apex:commandLink>`: This component creates a link that calls an action
- `<apex:actionPoller>`: This component periodically calls an action
- `<apex:actionSupport>`: This component makes an event (such as `onclick`, `onmouseover`, and so on) on another named component and calls an action
- `<apex:actionFunction>`: This component defines a new JavaScript function that calls an action
- `<apex:page>`: This component calls an action when the page is loaded

An action method can be called from the page using the `{!}` notation. For example, if your action method's name is `MyFirstMethod`, then you can use the `{!MyFirstMethod}` notation for calling the action method from the page markup.

 This action method can be from a standard controller or a custom controller or a controller extension.

A standard controller has a few standard action methods, as follows:

- `save`: This method inserts/updates a record. Upon successful completion it will be redirected to the standard detail page or a custom Visualforce page.
- `quicksave`: This method inserts/updates a record. There are no redirections to a detail page or custom Visualforce page.
- `edit`: This method navigates the user to the edit page for current record. Upon successful completion it will be returned to the page that invoked the action.
- `delete`: This method deletes the current record. It redirects the user to the list view page by selecting the most recently viewed list filter.

- `cancel`: This method cancels an edit operation. Upon successful completion it will be returned to the page that invoked the edit action.
- `list`: This method redirects to the list view page by selecting the most recently-viewed list filter.

For example, the following page allows us to insert a new customer or update an existing customer record. If we are going to use this page to update a customer record, then the URL must be specified with the ID query string parameter. Every standard controller has a getter method that returns the record specified by the ID query string parameter in the page URL. When we click on **Save**, the `save` action is triggered on the standard controller, and the details of the customer are updated. If we are going to use this page to insert a customer record, then the URL must not be specified as a parameter. In this scenario, when we click on **Save**, the `save` action is triggered on the standard controller, and a new customer record is inserted.

```
<apex:page standardController="Customer__c">
  <apex:form >
    <apex:pageBlock title="New Customer" mode="edit">
      <apex:pageBlockButtons >
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="My Content Section"
columns="2">
        <apex:inputField value="{!Customer__c.Name}"/>
        <apex:inputField value="{!Customer__c.Email__c}"/>
        <apex:inputField value="{!Customer__c.Address__c}"/>
        <apex:inputField value="{!Customer__c.Telephone__c}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```



The page markup allows you to access fields of a particular sObject by using `{!sObjectAPIName.FieldAPIName}`. For example, if you want to access the Email field of the Customer object, the page that uses the `Customer__c` standard controller can use `{!Customer__c.Email__c}` to return the value of the Email field of the customer who is in the current context.

The following page allows us to view a customer record. In this page also, the URL must be specified in the ID query string parameter. The getter method of the `Customer__c` standard controller returns the record specified by the ID query string parameter in the page URL:

```
<apex:page standardController="Customer__c">
  <apex:form >
    <apex:pageBlock title="Customer" mode="edit">
      <apex:pageBlockButtons >
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="Customer Details"
columns="2">
        <apex:outputField value="{!Customer__c.Name}"/>
        <apex:outputField value="{!Customer__c.Email__c}"/>
        <apex:outputField value="{!Customer__c.Address__c}"/>
        <apex:outputField value="{!Customer__c.
Telephone__c}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```



To check the accessibility of a particular object for the logged user, you can use the `{!$ObjectType.objectname.accessible}` notation. This expression returns a Boolean value. For a example, if you want to check the accessibility of the `Customer` object, you can use `{!$ObjectType.Customer__c.accessible}`.

```
<apex:page standardController="Customer__c">
  <apex:form >
    <apex:pageBlock title="New Customer" mode="edit">
      <apex:pageBlockButtons >
        <apex:commandButton rendered="{!$ObjectType.
Customer__c.accessible}" action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="Customer Details"
columns="2">
        <apex:inputField value="{!Customer__c.Name}"/>
        <apex:inputField value="{!Customer__c.Email__c}"/>
        <apex:inputField value="{!Customer__c.Address__c}"/>
        <apex:inputField value="{!Customer__c.Telephone__c}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```


The preceding code explains the usage of object accessibility. According to the example, you can see the **Save** button, only if the particular user has security permission to access the customer record.

Standard list controllers

The second controller type is the standard list controller which can be used for displaying or performing an action on a set of records (including related lists, list pages, and mass action pages). It allows us to filter records on a particular page. We can use standard list controllers for Account, Asset, Campaign, Case, Contact, Contract, Idea, Lead, Opportunity, Order, Product2, Solution, User, and all the custom objects.

How to use a standard list controller with Visualforce

Similar to the standard controller, we can specify the `standardController` attribute of the `<apex:page>` component. Additionally, we need to specify the `recordSetVar` attribute of the `<apex:page>` component.

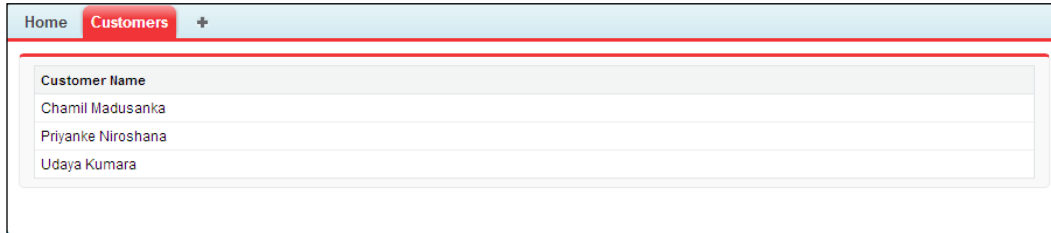


The `standardController` attribute specifies the type of records that we want to access. The `recordSetVar` attribute indicates that the page uses a list controller and the variable name (used to access data in the record collection) of the record collection.

The following markup explains how the page can access a list of records when the page is associated with a list controller. In the following example, you can refer to a list of customer records.

```
<apex:page standardController="Customer__c" recordSetVar="customers"
  sidebar="false">
  <apex:pageBlock >
    <apex:pageBlockTable value="{!customers}" var="a">
      <apex:column value="{!a.name}"/>
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>
```

The following screenshot illustrates the result of the preceding code:



The result page of the customer list example

Standard list controller actions

All the standard Visualforce components that have the `action` attribute can be used with a Visualforce page with a standard list controller. The usage of those components is same as for a standard controller. The following action methods are supported by all standard list controllers:

- `save`: This action method inserts/updates a record. Upon successful completion it will be redirected to the standard detail page or custom Visualforce page.
- `quicksave`: This method inserts/updates a record. There are no redirections to a detail page or a custom Visualforce page.
- `List`: This method redirects to the list view page by selecting the most recently viewed list filter when the filter ID is not specified by the user.
- `cancel`: This method cancels an edit operation. Upon successful completion it will be returned to the page which invoked the edit action.
- `first`: This method displays the first page of records in the set.
- `last`: This method displays the last page of records in the set.
- `next`: This method displays the next page of records in the set.
- `previous`: This method displays the previous page of records in the set.

List views in Salesforce standard pages can be used for filtering records that are displayed on the page. For example, on the customer home page, you can select **start with c view** from the list view dropdown and view the customers whose name starts with the letter c. You can implement this functionality on a page associated with a list controller.

Pagination can be added to a page associated with a list controller. The pagination feature allows you to implement the next and previous actions.

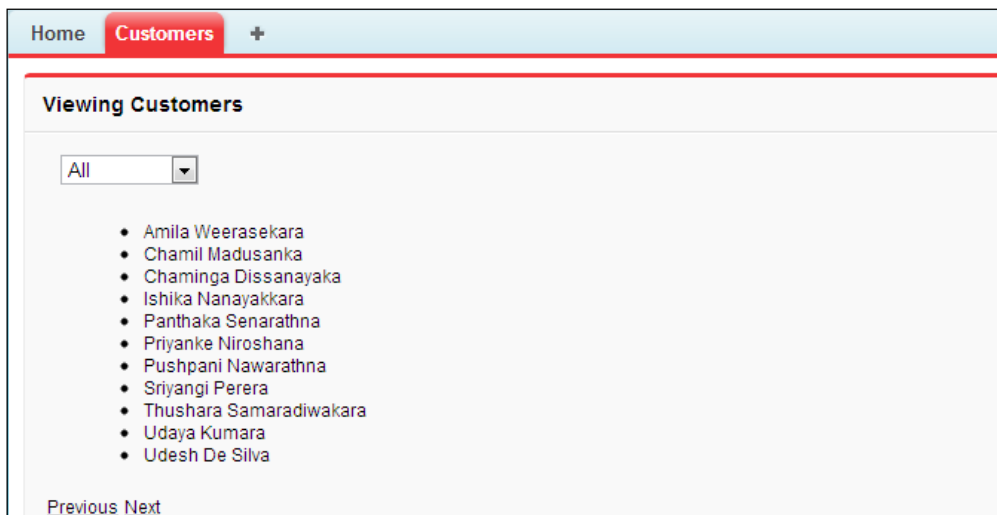
For example, to create a simple list of customers with a list view and pagination, create a page with the following markup:

```
<apex:page standardController="Customer__c" recordSetvar="customers">
  <apex:form id="theForm">
    <apex:pageBlock title="Viewing Customers">
      <apex:pageBlockSection >
        <apex:selectList value="{!filterid}" size="1">
          <apex:selectOptions value="{!listviewoptions}"/>
          <apex:actionSupport event="onchange"
rerender="list"/>
        </apex:selectList>
      </apex:pageBlockSection>


      <apex:pageBlockSection id="list">
        <apex:dataList var="a" value="{!customers}" type="1">
          {!a.name}
        </apex:dataList>
      </apex:pageBlockSection>

      <apex:panelGrid columns="2">
        <apex:commandLink action="{!previous}"
rerender="list">Previous</apex:commandlink>
        <apex:commandLink action="{!next}"
rerender="list">Next</apex:commandlink>
      </apex:panelGrid>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

The result of the preceding code is shown in the following screenshot:



Viewing customer list with pagination

 By default, a list controller returns 20 records per page. To control the number of records displayed on each page, use a controller extension to set the `pageSize` attribute.


Custom controllers and controller extensions

Custom controllers are used to implement the logic and functionality without using a standard controller and controller extensions are used to extend the logic and functionality of a standard controller or a custom controller. Custom controllers and Controller extension are written using Apex.

Understanding custom controllers

Custom controllers are used to implement logic and functionality without using a standard controller. Custom controllers are written using Apex. The following are the instances where you might want to use use a custom controller:

- Implement a completely different functionality without relying on the standard controller's behavior
- Override existing functionality
- Make new actions for the page
- Customize the navigation
- Use HTTP callouts or web services
- Use a wizard
- Have a greater control over accessing information on a page
- Run your page without applying permissions


 Only one controller can be used in a particular page.

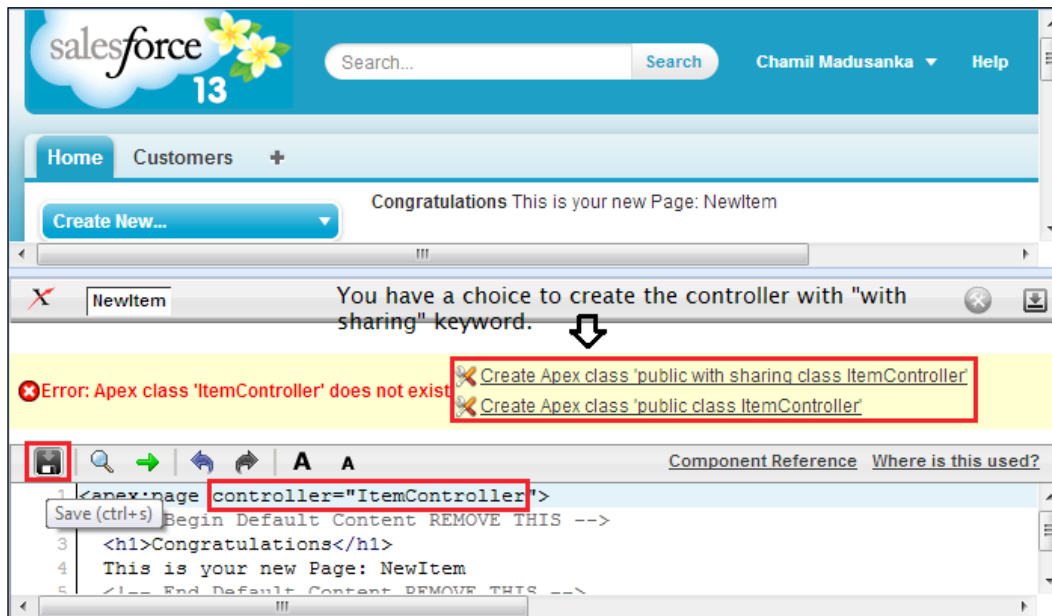
Building a custom controller

You can build a custom controller via the **Setup** page and the Visualforce editor. All the administrator and developer functionality are included in the **Setup** page, and you can find the **Setup** page from the menu which appears after clicking on your name (at the top of the page).

The Visualforce editor allows us to edit the markup of a Visualforce page in the same window and we can see that the result of the page will also be displayed on the same page. This editor has important functionality such as autocompletion, syntax highlighting, quick fix features (developers can create components on the fly), and compile on save using the following methods:

- Via the **Setup** page: This can be done by navigating to **Your Name | Setup | Develop | Apex Classes | New**.
- Via the Visualforce editor: After creating the page you can specify the custom controller's name in the controller attribute of the `<apex:page>` tag and then click on the **Save** button. Then, if you are a developer, the page will be asking you to create the class with the name that you entered. Then, the newly-created controller will be shown on the Visualforce editor, as shown in the next screenshot.

 You have the choice to write controller classes using the sharing or without sharing keyword, which is influenced to run the particular page in the system mode or user mode.



Creating a custom controller via the Visualforce editor

The following class is an example of custom controllers. This custom controller has the functionality for retrieving the existing item list from the `Item__c` custom object and adding a new item record. `insertNewItem` is the action method of `ItemController`. `ExistingItems` is a list of item properties which is used to retrieve the existing item records. The `ExistingItems` property has an overridden `get` method:

```
public with sharing class ItemController {
    //public item property for new insertion
    public Item__c NewItem{get;set;}
    public ItemController(){
        NewItem = new Item__c();
    }

    //get existing items to show in a table
    public List<Item__c> ExistingItems{
        get{
            ExistingItems = new List<Item__c>();
            ExistingItems = [SELECT Id, Name, Item_Name__c, Unit_
Price__c FROM Item__c LIMIT 100];
            return ExistingItems;
        }
        set;
    }

    public PageReference insertNewItem() {
        try{
            insert NewItem;
            //reset public property for new insert
            NewItem = new Item__c();
        }catch(DmlException ex){
            ApexPages.addMessages(ex);
        }
        return null;
    }
}
```



A custom controller uses a nonparameterized constructor. You cannot create a constructor that includes parameters for a custom controller.

The preceding controller is associated with the following Visualforce page. This page has two `<apex:pageBlock>` components: one for displaying the existing item records table and other for inserting new items:

```
<apex:page controller="ItemController">
  <apex:form >
    <apex:pageBlock title="Existing Items">
      <apex:pageBlockTable value="{!ExistingITems}"
var="oneItem" rendered="{!ExistingITems.size > 0}">
        <apex:column value="{!oneItem.Item_Name__c}"/>
        <apex:column value="{!oneItem.Unit_Price__c}"/>
      </apex:pageBlockTable>
      <apex:outputText value="No records to display"
rendered="{!ExistingITems.size == 0}"></apex:outputText>
    </apex:pageBlock>
    <apex:pageBlock title="New Item">
<apex:pageMessages ></apex:pageMessages>
      <apex:pageBlockSection >
        <apex:inputField value="{!NewItem.Item_Name__c}"/>
        <apex:inputField value="{!NewItem.Unit_Price__c}"/>
      </apex:pageBlockSection>
      <apex:pageBlockButtons >
        <apex:commandButton action="{!insertNewItem}"
value="save"/>
      </apex:pageBlockButtons>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```


Understanding controller extension

Controller extensions are used to extend the logic and functionality of a standard controller or a custom controller. A controller extension cannot be on a page without a standard controller or a custom controller. Controller extensions are written using Apex. Use controller extensions when you want to:

- Keep the majority of functionality of a standard or custom controller as it is and add more functionality
- Build a Visualforce page that should run according to the user's permissions

Building a controller extension

We can build a controller extension in the same way as for building the custom controller.


 Extensions cannot live by themselves on a page. They can be used on a Visualforce page with a custom controller or a standard controller.

The following class is a simple example of a controller extension. This controller extension is used to extend the logic and the functionality of the `Order__c` custom object's standard controller. In this extension, we have a one-parameterized constructor to fetch the order record from the standard controller. `getRecord()` is the method for fetching records from the standard controller. The `prepareFullOrder()` method is a custom method that is implemented for querying the order lines of a particular order:

```
public with sharing class OrderViewExtension{
    public Order__c CurrentOrder{get;set;}
    public List<Order_Line__c> OrderLines{get;set;}

    public OrderViewExtension(ApexPages.StandardController controller)
    {
        CurrentOrder = new Order__c();
        this.CurrentOrder = (Order__c)controller.getRecord();
        prepareFullOrder();
    }

    public void prepareFullOrder(){
        OrderLines = new List<Order_Line__c>();
        OrderLines = [SELECT Id, Name, Price__c, Item__c, Item__r.
Unit_Price__c,Item__r.Item_Name__c, Order__c, Quantity__c FROM Order_
Line__c WHERE Order__c =: this.CurrentOrder.Id];
    }
}
```

 A controller extension uses one-parameterized constructor with the `ApexPages.StandardController` type of argument or a custom controller type.

The following Visualforce page uses the preceding controller extension. On the page, we have a page block with two sections. The first section shows us the order header details. The second section is there to show the order lines of a particular order:

```
<apex:page standardController="Order__c" extensions="OrderViewExtension">
  <apex:form >
    <apex:pageBlock >
      <apex:pageBlockSection title="Order Header">
        <apex:outputField value="{!Order__c.Name}"/>
        <apex:outputField value="{!Order__c.Customer__c}"/>
        <apex:outputField value="{!Order__c.Planned_Delivery_
Date__c}"/>
      </apex:pageBlockSection>
      <apex:pageBlockSection title="Order Lines" columns="1">
        <apex:pageBlockTable value="{!OrderLines}" var="line">
          <apex:column value="{!line.Name}"/>

          <apex:column headerValue="Item">
            <apex:outputLink value="/{!line.Item__c}"
target="_blank">{!line.Item__r.Item_Name__c}</apex:outputLink>
          </apex:column>
          <apex:column value="{!line.Item__r.Unit_Price__c}"/>
          <apex:column value="{!line.Quantity__c}"/>
          <apex:column value="{!line.Price__c}"/>
        </apex:pageBlockTable>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Controller methods

There are three types of methods which can be used within a custom controller or a controller extension:

- Getter methods
- Setter methods
- Action methods

Getter methods

Developers can use getter methods to display a database or other computed values in the Visualforce markup. This means that getter methods are used to pass data from Apex controllers to the Visualforce page. There are two ways to define getter methods.

Typically, getter methods are named as `getVariable`, where the variable is the name of the attribute that is returned by the getter method:

```
public class GetterSetterExample{
    String GetterVariable;

    public String getGetterVariable() {
        return GetterVariable;
    }
}
```

A getter method can define an attribute by using the default getter and setter methods:

```
public class GetterSetterExample{
    public String GetterVariableDefault{get;set;}
}
```

The variable can be accessed on the Visualforce page with the `{!}` expression.

Setter methods

Setter methods are used to pass user-defined values to the Apex controller. Setter methods are defined in the same way as getter methods are defined. The following example uses default getter and setter methods to search for items that are already in the database:

```
public with sharing class SearchItemController {

    public List<Item__c> ExistingItems{get;set;}
    public String Keyword{get;set;}

    public SearchItemController(){
        ExistingItems = new List<Item__c>();
    }
    public void SearchItems(){
        ExistingItems = [SELECT Id, Name, Item_Name__c, Unit_Price__c
FROM Item__c WHERE Item_Name__c LIKE: ('%'+Keyword+'%')];
    }
}
```

The following is the Visualforce page that uses the preceding controller. The `Keyword` attribute has the default getter and setter methods for the `<apex:inputText>` component, which is used to acquire the user's input. The `ExistingItems` list attribute also has the default getter and setter methods to search and display the search result. When the user enters a keyword to search for and clicks on the **Search** button, the `SearchItems()` action method will be executed and this will acquire the keyword search text and run the query to search for the items. Before the action method executes, the keyword setter method will be executed. Then the query result will be collected to the `ExistingItems` list attribute and then the `ExistingItems` getter method will be executed and the page will display the search result:

```
<apex:page controller="SearchItemController">
  <apex:form >
    <apex:pageBlock >
      <apex:pageBlockSection >
        <apex:pageBlockSectionItem >
          <apex:outputLabel value="Item Name Or keyword"></
apex:outputLabel>
          <apex:inputText value="{!Keyword}"/>
        </apex:pageBlockSectionItem>
        <apex:commandButton value="Search" action="{!SearchItems}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>


    <apex:pageBlock title="Search Result" id="searchResult">
      <apex:pageBlockTable value="{!ExistingItems}"
var="oneItem" rendered="{!ExistingItems.size > 0}">
        <apex:column value="{!oneItem.Item_Name__c}"/>
        <apex:column value="{!oneItem.Unit_Price__c}"/>
      </apex:pageBlockTable>
      <apex:outputText value="No records to display"
rendered="{!ExistingItems.size == 0}"></apex:outputText>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Action methods

Action methods are used to implement our custom or extended logic and functionality in a custom controller or a controller extension. Action methods can be triggered on page events such as button clicks or JavaScript events. In Visualforce pages, we can define the action attribute in many standard Visualforce components. The components are `<apex:commandButton>`, `<apex:commandLink>`, `<apex:actionPoller>`, `<apex:actionSupport>`, `<apex:actionFunction>`, and `<apex:page>`. The preceding item search example has an action method called `SearchItems`. `SearchItems` is used to query items according to the user input given for item search.

Working with large sets of data on the Visualforce page

On a Visualforce page, we have to work with a single record as well as large sets of data. When we work with large sets of data, we may use iteration components such as `<apex:pageBlockTable>`, `<apex:repeat>`, and `<apex:dataTable>`. These iteration components are limited to a maximum of 1000 items in a collection. Refer the search item example for the usage of the iteration component. We have used `<apex:pageBlockTable>` in the previous search item example.

[ Custom controllers and controller extensions adhere to the Apex governor limits.]

Visualforce provides the "read-only mode" feature to overcome the limit on the number of rows that can be queried within one request and the limit on the number of collection items that can be iterated on the page. There are two ways to set up the Visualforce's read-only mode feature, which are as follows:

- **Setting the read-only mode for controller methods:** For this setting, we can define Visualforce controller methods with the `@ReadOnly` annotation. This read-only mode relaxes the number of records queried within one query from 50,000 to 1 million rows. The `@ReadOnly` annotation for the read-only mode is used in JavaScript remoting as the target of remote JavaScript call to load the data set for the `<apex:chart>` component and display some values in a component.

- **Setting the read-only mode for an entire page:** This read-only mode can be enabled by adding a `true` value for the `readOnly` attribute, which is on `<apex:page>`. This read-only mode relaxes the number of records queried within one query from 50,000 to 1 million. It also increases the maximum number of items in a collection for an iteration component. Because this is a read-only mode, you have to note that the page cannot execute any DML operation.

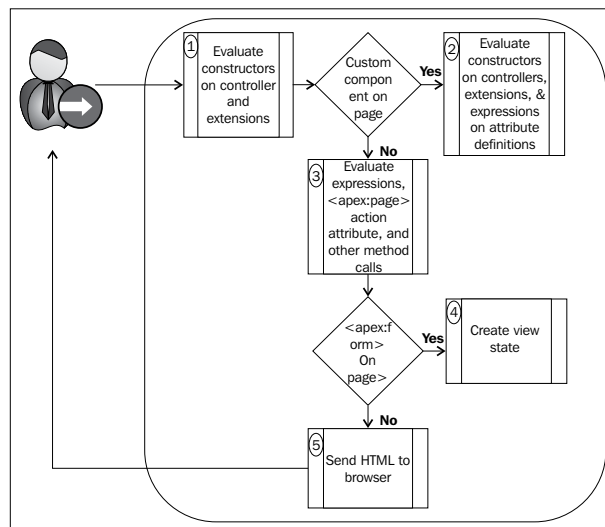
Order of execution of a Visualforce page

A Visualforce page has a life cycle or life-time. This time is defined as the period between the creation of the page and its destruction during the user session. The life cycle is defined by the type of Visualforce page request and the content of the page. There are two types of Visualforce page requests, which are as follows:

- Get request
- Postback request

Order of execution for a Visualforce page's get requests

When we request a new page by entering a URL or by clicking on a button or a link, a get request is created. The following diagram illustrates how a Visualforce page interacts with a custom controller or a controller extension during a get request:



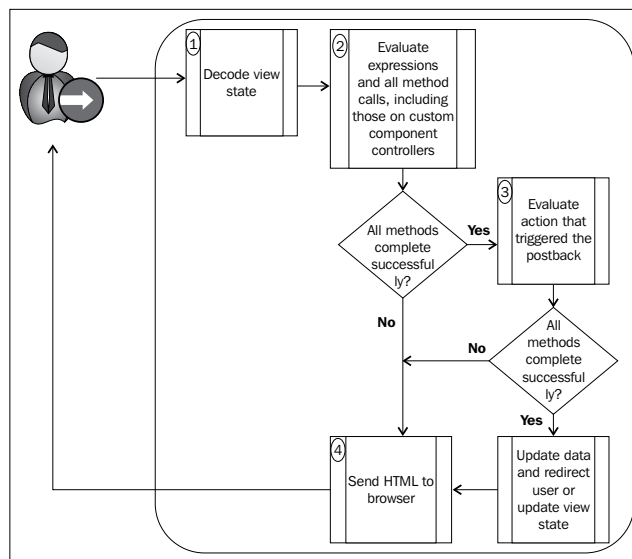
Order of Execution for Visualforce page's get requests

The order of execution is as follows:

1. Constructor methods are called by initiating the controller objects.
2. If there are any custom components, they are created and constructor methods are called on their associated class. If any attribute is specified in a component using an expression, those expressions are also evaluated.
3. Any `assignTo` attributes and expressions are evaluated. After that, the `action` attribute on the `<apex:page>` component is evaluated and all the getter or setter methods are called.
4. If the page contains an `<apex:form>` tag, then all of the information representing the state of the database is encrypted and saved in the view state between page requests. Whenever the page is updated, that view state is also updated.
5. Finally, the resultant HTML is sent to the browser. If there are any client-side technologies (such as JavaScript, and CSS), the browser executes them.

Order of execution for a Visualforce page's postback requests

Some user interactions (for example, a save action triggered by the user's button click) require page updates, typically those page updates are performed by postback requests. The following diagram illustrates how a Visualforce page interacts with a custom controller or a controller extension during a postback request:



Order of execution for a Visualforce page's postback requests

The order is as follows:

1. The view state is decoded and used as the basis for updating the values on the page during a postback request.
2. Expressions are evaluated and setters are executed.
3. The action is executed. On its successful completion, the data is updated. If the postback request redirects the user to the same page, the view state is updated.
4. The results are sent to the browser.



If we want to execute an action without performing validations on the input or data changes on the page, we can use an immediate attribute with the `true` value for a particular component.

The postback request can end with a page redirect and sometimes the custom controller or the controller extension may be shared on both the originating page and the redirected page. If the postback request contains an `<apex:form>` component, only the ID query parameter is returned.



The `action` attribute of the `<apex:page>` component is evaluated only during a get request. Once the user is redirected to another page, the view state and controller objects are deleted.

Validation rules and standard controllers/ custom controllers

Validation rules can be applied to custom or standard objects for validating data on insert and update operations. When we perform such operations on a Visualforce page, it uses a standard controller or a custom controller, and that record may cause a validation rule error, which we can display on the Visualforce page as we do on standard pages. A validation rule has two options to select the position for displaying the error for a particular field. If we choose **top of the page**, the error can be displayed by using the `<apex:pageMessages>` or `<apex:messages>` component within the `<apex:page>` component. If we choose the **field** option, the error will be shown in the associated field residing next to the `<apex:inputField>` component. For an example, you can see the sample page given in the *Building a custom controller* section.

You can try the example by entering a non-numeric character for the **Unit Price** field. An error message will be displayed near to the `Unit_Price__c` field, related to the `<apex:inputField>` component.

Using the transient keyword

The `transient` keyword is used for declaring variables, and is used in Apex classes. Declaring a variable as `transient` reduces the view state size. Variables with the `transient` keyword cannot be saved and should not be transmitted as a part of the view state of the particular Visualforce page. Transient variables are needed only for the duration of a page request.

The `transient` keyword is used in a serializable Apex class, which means the classes that implement the `Batchable` or `Schedulable` interfaces. The following Apex objects are natively considered as `transient`:

- `PageReference`
- `XmlStreamClasses`
- Collections (only if the type of object that they hold is automatically marked as `transient`)
- Most objects generated by system methods such as `Schema.getGlobalDescribe`
- Static variables
- Instances of the `JSONParser` class

The following example has a `transient datetime` variable and a non-`transient datetime` variable. This example shows the major feature of `transient` variables, which is that they cannot be saved and should not be a part of the view state. When we click on the **Refresh** button, the `transient` date will be recreated but the non-`transient` date will have its original value:

```
<apex:page controller="TransientExampleController">
  Non Transient Date: {!t1} <br/>
  Transient Date    : {!t2} <br/>
  <apex:form >
    <apex:commandLink value="Refresh"/>
  </apex:form>
</apex:page>

public with sharing class TransientExampleController {
  DateTime t1;
  transient DateTime t2;
  public String getT1() {
    if (t1 == null) t1 = System.now();
  }
}
```



```
        return '' + t1;
    }

    public String getT2() {
        if (t2 == null) t2 = System.now();
        return '' + t2;
    }
}
```

Considerations for creating custom controllers and controller extensions

When you are creating custom controllers and controller extensions, keep the following consideration in mind:

- The most important thing to keep in your mind is Apex governor limits.
- Apex classes can be run in the system mode and user mode by using `without sharing` and `with sharing` respectively. Sensitive data can be exposed without sharing controllers.
- The `webservice` methods must be defined as `global`. All other methods are `public`.
- Try to access the database in less time by using sets, maps, or lists. This will increase the efficiency of your code.
- Apex methods and variables are not instantiated in a guaranteed order.
- You cannot implement **Data Manipulation Language (DML)** in the constructor method of a controller.
- You cannot define the `@future` annotation for any getter method, setter method, or constructor method of a controller.
- Primitive data types (String, Integer, and so on) are passed by value and non-primitive Apex data types (list, maps, set, sObject, and so on) are passed by the reference to a component's controller.

Summary

In this chapter, we became familiar with types of controllers and extensions. We learned the differences and the usage of standard controller, standard list controller, custom controller, and controller extension. We learned how to handle the code in order work with a large set of data. Further, we have seen the order of execution of a Visualforce page, usage of the `transient` keyword, and the interconnection between validation rules and controllers.

3

Visualforce and Standard Web Development Technologies

The combination of native Visualforce markup and standard web development technologies can be used to build rich UIs for Force.com applications. However HTML rendering of Visualforce is complicated and there are many ways to change Visualforce's generated default HTML by using additional resources such as CSS and JavaScript.

This chapter teaches how to develop Visualforce pages with standard web development technologies such as CSS, JavaScript, jQuery, and HTML5. The usage of static resources for CSS, JavaScript, and jQuery are also included in the chapter. The following topics will be covered in this chapter:

- Styling Visualforce pages
- Using JavaScript in Visualforce pages
- Using jQuery in Visualforce pages
- HTML5 and Visualforce pages

Let's build rich user interfaces.

Styling Visualforce pages

We use Visualforce pages to accomplish both simple UI requirements and sophisticated UI requirements. In the case of implementing sophisticated scenarios, we cannot meet such requirements by using only Salesforce's standard styles. We can customize the look and feel of a Visualforce page by using our own stylesheets or styles.

Many standard Visualforce components have the `style` and/or `styleClass` attribute. We can use either of these attributes to customize the page by using CSS. This allows us to change the default style (width, height, color, and font) of components. There are two types of styles in Visualforce, which are as follows:

- Salesforce styles
- Custom styles

Salesforce styles

Salesforce standard pages have standard styles that can be used on Visualforce pages. When we use standard Visualforce components such as `<apex:inputField>`, `<apex:pageBlock>`, `<apex:pageBlockTable>`, and `<apex:detail>`, they acquire the default styles provided by Salesforce. The `tabStyle` attribute in the `<apex:page>` or `<apex:pageBlock>` component can specify the style of a particular object tab. It will change the color scheme of the preceding components. When we use a standard controller, Visualforce page components inherit the styles of the associated objects. When we use a custom controller, we can use the styles of any of Salesforce's standard tabs by using the `tabStyle` attribute on the `<apex:page>` tag.

Custom styles

We can extend the Salesforce styles by using custom styles. Custom styles can be added to a Visualforce page by using the `style` and/or `styleClass` attribute. The `style` attribute and the `styleClass` attribute are available on most Visualforce components. The `style` attribute allows you to add inline custom CSS statements. The `styleClass` attribute allows you to add custom styles via a class name which is specified in a CSS file.

In the following example, the sample Visualforce page has custom styles that have been added via the `style` and `styleClass` attributes:

```
<apex:page >
<style>
    .sample {font-weight: bold;}
</style>
```

```
<apex:outputText value="This text is styled via style attribute"
style="font-weight: bold;"/> <br/>
<apex:outputText value="This text is styled via styleClass attribute"
styleClass="sample"/>
</apex:page>
```

In the preceding example, we have implemented CSS on the Visualforce page. But, if we use the same style in multiple locations, we have to add that particular CSS file to every Visualforce page. We can use static resources to overcome this problem. This is another way to bind custom styles to the Visualforce markup.

Static resources are uploaded to the Force.com platform via the **Setup** screen. The Force.com platform allows us to upload images, stylesheets, JavaScript, and archives (.zip and .jar files). The following is the stylesheet (CSS filename: `main.css`) we have used above Visualforce page:

```
.sample {font-weight: bold;}
```

Let's see how we can upload this stylesheet into static resources. The resource name is `CustomMainStyle` which must be unique. The resource can be created by navigating to the following path:

Your Name | Setup | Develop | Static Resources | New

Static Resource

Help for this Page ?

Static Resource Edit Save Cancel

Static Resource Information | = Required Information

Name

Description

File main.css

Cache Control ▼

Save Cancel

Chat

Use static resources to store CSS file

We can refer to the `CustomMainStyle` static resources in a Visualforce page as follows. The `<apex:stylesheet>` tag can be used to include a stylesheet. The resource name is used to refer the static resource in a Visualforce page as given in the following code:

```
<apex:page>
  <apex:stylesheet value="{!$Resource.CustomMainStyle}"/>

  <apex:outputText value="This text is styled via styleClass attribute
  via static resources" styleClass="sample"/>
</apex:page>
```



If you want to remove the Salesforce standard styles entirely, you have to set a false value to the `standardStylesheets`, `sidebar`, and `showHeader` attributes on the `<apex:page>` tag. If you stop loading standard Salesforce stylesheets, you can reduce the size of your Visualforce page:

```
<apex:page sidebar="false" showHeader="false"
  standardStylesheets="false">
</apex:page>
```

We have mentioned earlier that the Force.com platform allows us to upload archive files (such as ZIP and JAR files) as static resources. In such a scenario, a ZIP file can contain resources such as images, CSS files and JavaScript files. In this case, we can refer an individual resource within the ZIP file by using the `URLFOR` function. The `URLFOR` function has two parameters. The first parameter is the name of the static resource, which we provide while uploading a static resource. The second parameter is the path to the particular file within the ZIP file. The static resource that is used in the following example is a ZIP file that has the `main.css` stylesheet in a directory called `CustomStyleZipFolder`.

The screenshot shows the 'Static Resource Edit' interface in Salesforce. At the top right, there is a 'Help for this Page' link. Below the title, there are 'Save' and 'Cancel' buttons. The main section is titled 'Static Resource Information' and contains several fields: 'Name' with the value 'CustomStyleZip', 'Description' with the value 'Custom Style Zip file', 'File' with a 'Choose File' button and the filename 'CustomStyleZip.zip', and 'Cache Control' with a dropdown menu set to 'Public'. At the bottom of the form, there are 'Save' and 'Cancel' buttons. In the bottom right corner, there is a 'Chat' button with a speech bubble icon.

Upload zip file as a static resource

```

<apex:page >
<apex:stylesheet value="{!URLFOR($Resource.CustomStyleZip,'/
CustomStyleZipFolder/main.css')}" />

<apex:outputText value="This text is styled via styleClass attribute
via static resources" styleClass="sample"/> <br/>

<apex:outputText value="Following image is loaded via css class"/>
<br/>
<apex:outputPanel styleClass="imageCls">

</apex:outputPanel>

<apex:outputText value="Following image is loaded directly from static
resource"/> <br/>
<apex:image value="{!URLFOR($Resource.CustomStyleZip,'/
CustomStyleZipFolder/images/sfLogo.png')}" width="100" height="50"/>

</apex:page>

```

There is a special scenario with the static resources where you can use relative paths of files in static resource. This allows us to refer to other contents within the archive in a relative manner. For example, the `Main.css` file has the following style:

```

.sample {font-weight: bold;}
.imageCls {background:url(images/sfLogo.jpg) no-repeat top left;
width: 100px;
height: 100px;
display: block;
}

```

In the preceding style code, the image's path needs to be specified relatively to the `Main.css` file. In this scenario, we prepare our `CustomStyleZipFolder` directory, that contains the `Main.css` file and the `images` folder. Here, the `sfLogo.jpg` image present in `Images` is referred in `Main.css`.

Then we only need to include the `Main.css` file into a Visualforce page. We do not need to worry about the relative path in the stylesheet, because the static resource contains both the stylesheet and the image.



Maximum size of a single static resource is 5 MB. Maximum size of static resources that we can have in an organization is 250 MB.

Using JavaScript in Visualforce pages

JavaScript is one of the key browser technologies for Visualforce pages. JavaScript provides the framework for communicating between other JavaScript objects, HTML elements, and the Visualforce controller. We can use JavaScript libraries as well as some Visualforce components (such as `<apex:actionFunction>`, `<apex:actionSupport>`, `<apex:commandButton>`, `<apex:commandLink>`) with Visualforce pages. JavaScript code can be written in a Visualforce page and can be included in a Visualforce page by using a static resource. This is the best method to use to include a JavaScript library in a Visualforce page. We can use the `<apex:includeScript>` component to include a JavaScript library from static resources. For example:

```
<apex:includeScript value="{!$Resource.MyJSFile}"/>
```

Accessing Visualforce components in JavaScript

When we refer Visualforce components in the JavaScript code, the ID attribute comes into play. Every Visualforce component has an ID attribute. The ID attribute must be specified with a particular component to refer it in JavaScript, and it is used to bind the two components together. When the page is rendered, this ID attribute is a part of DOM ID for a particular component. The ID attribute must be unique as well. The following code snippet shows a way of binding two components using the `id` attribute:

```
<apex:outputLabel value="Label Name" for="item"/>
<apex:inputField id="item" value="{!item__c.Name}"/>
```

The following example provides an idea about the way to handle JavaScript in a Visualforce page. This page has been implemented to change the pick list value by checkbox through JavaScript. The JavaScript code is included within the `<script>` tag. The JavaScript function has two arguments. The first argument is the element that triggered the event (`input`) and the second one is the DOM ID (`id`) of the target pick list field. The `{!$Component.inputStatus}` expression obtains the DOM ID of the HTML element generated by the `<apex:inputField id="inputStatus" value="{!order.Status__c}"/>` component:

```
<apex:page controller="OrderStatusUpdate" id="pageId">
  <script type="text/javascript">
    function updateStatus(input,id) {
      if(input.checked){
        document.getElementById(id).value="Processing";
        //alert(document.getElementById(id).value);
      }else{
```

```

        document.getElementById(id).value="New";
        //alert(document.getElementById(id).value);
    }
}
</script>
<apex:form id="formId">
<apex:pageBlock id="pageBId">
    <apex:pageBlockTable id="tableId" value="{!Orders}"
var="order">
        <apex:column value="{!order.Name}"/>
        <apex:column value="{!order.Customer__c}"/>
        <apex:column id="checkId" headerValue="Status">
            <apex:inputField id="inputStatus" value="{!order.
Status__c}" />
        </apex:column>
        <apex:column headerValue="Started Processing" >
            <apex:selectCheckboxes onclick="updateStatus(this, '{!$
Component.inputStatus}');" >

                </apex:selectCheckboxes>
            </apex:column>

        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:form>

</apex:page>

```

The following code is the associated controller class for the preceding Visualforce page. It retrieves the existing orders as follows:

```

public with sharing class OrderStatusUpdate {

    public List<Order__c> Orders{get;set;}

    public OrderStatusUpdate(){
        Orders = new List<Order__c>();
        Orders = [SELECT id, Name, Customer__c, Status__c, Planned_
Delivery_Date__c, Delivered__c FROM Order__c LIMIT 1000];
    }
}

```



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

JavaScript remoting for Apex controllers

JavaScript remoting is the process that provides support for some methods in APEX controllers that are to be called via JavaScript. This feature allows us to implement complex and dynamic behaviors that cannot be accomplished using standard Visualforce Ajax components. JavaScript remoting was released as a Developer Preview in Spring '11. Since the Summer '11 release, JavaScript remoting provides support for additional return data types. Also, the references to the same objects are no longer duplicated in the response. JavaScript remoting has three main parts:

- The JavaScript code which is used to invoke a remote method
- The remote method in the Apex controller
- The callback function (written in JavaScript) in a Visualforce page

To use JavaScript remoting, your request must take the following form:

```
[<namespace>.<controller>.<method> ([params...],  
<callbackFunction>(result, event)  
{  
  // callback function logic  
}, {escape:true});
```

The description of the preceding code is as follows:

- `namespace`: This is your organization's namespace. This is only required if the class comes from an installed package.
- `controller`: This is the name of your Apex controller.
- `method`: This is the name of the Apex method you're calling.
- `params`: This is a comma-separated list of parameters that your method takes.
- `callbackFunction`: This is the name of the function that handles the response from the controller. It returns the status of the call and the method result.
- `escape`: This specifies whether your response should be escaped (by default, `true`) or not (`false`).

The remote method must begin with the `@RemoteAction` annotation as follows:

```
@RemoteAction  
global static String getItemId(String objectName) { ... }
```

The remote method can have the following data types as arguments:

- Apex primitives (String, Integer, and so on)
- Collections (Set, List, Map)
- sObject (Standard objects and custom objects)
- User-defined Apex classes and interfaces

The remote method can return the following data types:

- Apex primitives (String, Integer, and so on)
- sObjects (standard objects and custom objects)
- Collections (Set, List, Map)
- User-defined Apex classes and enums
- SelectOption
- PageReference
- SaveResult
- UpsertResult
- DeleteResult



The remote method must be uniquely identified by the name and number of parameters. For example, we cannot write a remote method with the same method name and equal number of arguments and different type of arguments.

The following example shows how to use JavaScript remoting in a Visualforce page and the Apex controller:

```
<apex:page controller="JavaScriptRemotingController" id="pageId">
  <script type="text/javascript">
    function updateStatus(input,id) {
      var inputStatus=id;
      JavaScriptRemotingController.doStartShipping(inputStatus,function(res
      ult,event){

    }, {escape:true});
  }</script>

  <apex:form id="formId">
    <apex:pageBlock id="pageBId">
```

```
        <apex:pageBlockTable id="tableId" value="{!Orders}"
var="order">
        <apex:column value="{!order.Name}"/>
        <apex:column value="{!order.Customer__c}"/>
        <apex:column id="checkId" headerValue="Status">
            <apex:inputField id="inputStatus" value="{!order.
Status__c}" />
        </apex:column>
        <apex:column headerValue="Started Processing" >
            <apex:selectCheckboxes onclick="updateStatus(this, '{!
order.Id}')";">
        </apex:selectCheckboxes>
        </apex:column>

        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:form>
</apex:page>
```

This is the associated Apex controller:

```
global with sharing class JavaScriptRemotingController {

    public List<Order__c> Orders{
        get{
            Orders = new List<Order__c>();
            Orders = [SELECT id, Name, Customer__c, Status__c, Planned_
Delivery_Date__c, Delivered__c FROM Order__c LIMIT 1000];
            return Orders;
        }
        set;
    }

    public JavaScriptRemotingController(){

    }

    @RemoteAction
    global static Item__c doStartShipping(String para){
        Order__c updateOrder;
        try{
            updateOrder=[SELECT id, Name, Customer__c, Status__c, Planned_
Delivery_Date__c, Delivered__c FROM Order__c Where Id =: para];
            updateOrder.Status__c = 'Shipping';
            update updateOrder;
        }
    }
}
```

```

    } catch (DMLException e) {
        ApexPages.addMessages(e);
        return null;
    }
    return null;
}
}

```

Using jQuery in Visualforce pages

jQuery is an open source JavaScript library which allows us to implement client-side scripting of HTML. jQuery has been designed to be capable of extending the main libraries with new plugins for introducing a wide variety of new features. And also it allows us to navigate a document, select a DOM element, create animations, event handling, and develop Ajax applications.

When we develop Visualforce pages, jQuery can be used to simplify the UI developments. For example, jQuery is used to simplify the DOM manipulations and give access to the library of UI elements, and simplify the Ajax techniques and technologies of mobile devices.

The following example shows the jQuery version of our previous example. This is used to explain the JavaScript code in Visualforce. This example uses the `Order__c` standard controller. This page needs the ID parameter with an order id:

<https://c.ap1.visual.force.com/apex/JQueryExample?id=a02900000086H1r>

This page renders the order detail page. We are using jQuery to fulfill our requirement. Therefore, the Visualforce page needs to include the jQuery library for jQuery implementations. In the following example we used an online reference of the main jQuery library:

```

<apex:includeScript value="https://ajax.googleapis.com/ajax/libs/
jquery/1.7.2/jquery.min.js" />

```

You can also use static resources to include the jQuery library. The usage is same as in the JavaScript and CSS examples.



There are other JavaScript libraries with the same default global variable name (\$). If we also use the same global variable name, there will be a conflict at the client side. Our jQuery functions will not work. To eliminate that conflict, we can use `jQuery.noConflict()` and assign it to another global variable and use that new global variable in our jQuery code.

```
<apex:page StandardController="Order__c" id="pageId">
  <apex:includeScript value="https://ajax.googleapis.com/ajax/libs/
jquery/1.7.2/jquery.min.js" />

  <script type="text/javascript">
    jQuery = jQuery.noConflict();
    jQuery(document).ready(function() {
      jQuery('.checkBox').click(function () {
        jQuery('.inputStatus').val('Processing');
      });
    });
  </script>
  <apex:form id="formId">
    <apex:pageBlock id="pageBId">
      <apex:pageBlockSection id="pBlockSection">
        <apex:outputField value="{!Order__c.Name}"/>
        <apex:outputField value="{!Order__c.Customer__c}"/>
        <apex:inputField styleClass="inputStatus"
value="{!Order__c.Status__c}" />
        <apex:pageBlockSectionItem id="pbSectionItem">
          <apex:outputLabel value="Mark as Started
Processing"></apex:outputLabel>
          <apex:selectCheckboxes styleClass="checkBox" >
            </apex:selectCheckboxes>
          </apex:pageBlockSectionItem>
        </apex:pageBlockSection>
      </apex:pageBlock>
    </apex:form>
  </apex:page>
```



If we don't use an id attribute for a particular component, Visualforce uses a dynamically-generated id, for example, j_id0, j_id0:j_id1. Consider an example, we have specified the id attribute for `<apex:inputField id="inputOne"/>`. But we haven't specified any id attribute for parent components of inputOne. We can select such a component using jQuery. It is called partial selectors. For example: `jQuery('[id*= inputOne]')`

HTML5 and Visualforce pages

HTML5 is the new standard of HTML. The previous version of HTML is HTML 4.01. HTML5 has new features such as new elements, new attributes, video and audio support, 2D/3D graphic support, full CSS3 support, local storage, local SQL database support, and featuring web applications. With these features, we can reduce the use of external plugins. There are also more markups to replace scripting. HTML5 has better error handling mechanism.

When it comes to the Force.com developments, we can use HTML5 for Visualforce page developments and develop mobile web applications. In the Force.com platform, HTML5 plays a major role in developing web-based mobile applications. For example, recently Salesforce has released touch.salesforce.com, which uses HTML5.

By default, Visualforce pages are functioned with `docType` of HTML 4.01 transitional. Since the Winter '12 version, Visualforce pages are supported to change the `docType` attribute in the `<apex:page>` tag. In a pure HTML5 page, the `<!DOCTYPE html>` tag must be specified at the top of the page. The `docType` attribute of `<apex:page>` achieves that requirement.

The following is the example usage for the `docType` Visualforce attribute on the `<apex:page>` component:

```
<apex:page docType="html-5.0"><!-- HTML5 --></apex:page>
<apex:page docType="html-4.0.1-transitional"><!-- HTML 4.0.1
Transitional --></apex:page>
<apex:page docType="xhtml-5.0.1-strict"><!-- XHTML 5.0.1 Strict--></
apex:page>
```

The following example is a Visualforce page with drag-and-drop functionality by using HTML5. Here we have a rectangle and an image which is referring from static resources. We can drag the image into the rectangle:

```
<apex:page docType="html-5.0" sidebar="false" showHeader="false"
standardStylesheets="false" cache="true" >

<html>
<head>
<style type="text/css">
#div1 {width:400px;height:400px;padding:10px;border:1px solid
#aaaaaa;}
</style>
<script>
function allowDrop(ev)
{
ev.preventDefault();
```

```
}

function drag(ev)
{
ev.dataTransfer.setData("Text",ev.target.id);
}

function drop(ev)
{
ev.preventDefault();
var data=ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<p>Drag the Salesforce logo into the rectangle:</p>

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
<br/>


</body>
</html>
</apex:page>
```

Summary

In this chapter we became familiar with the combined usage of native Visualforce markup and standard web development technologies. We have seen the way to build rich UIs in the Force.com platform by using CSS, JavaScript, jQuery, and HTML5. We have learned the usage of static resources for CSS, JavaScript, and jQuery.

4

Visualforce Custom Components

Salesforce has a large collection of standard Visualforce components such as `<apex:detail>`, `<apex:pageBlock>`, `<apex:pageBlockTable>`, and `<apex:relatedList>`. They are ready to be used in Visualforce pages, and the Force.com platform allows us to build our own Visualforce components which can be used in Visualforce pages.

This chapter serves as an overview of Visualforce custom components and further explains how to create a custom component. This chapter covers the following topics:

- Understanding Visualforce custom components
- How to create and use a custom component in a Visualforce page
- Custom attributes and custom controllers

Let's build our own Visualforce custom components.

Understanding Visualforce custom components

There are lots of standard Visualforce components (such as `<apex:detail>`, `<apex:pageBlock>`, `<apex:pageBlockTable>`, and `<apex:relatedList>`) which can be reused in Visualforce pages. A standard Visualforce component is a pre-built, encapsulated code segment. These standard Visualforce components are built according to common usage.

The Force.com platform allows us to develop custom Visualforce components that can be reused within a particular application. Custom components can be developed using both Apex and Visualforce. For example, suppose we want to create a customer summary with recent orders and we need to use this functionality in different locations in our order processing app. We also need to specify the number of recent orders. According to the specified number, the number of recent orders displayed in the customer's summary will be changed. The use of Visualforce custom components becomes the best choice for implementing such a specific requirement.

A Visualforce custom component can have zero or more attributes to pass as parameters into the component. A custom component with attributes is like a parameterized Apex method. We can change the value of an attribute during the final usage level (in a Visualforce page).

Creating and using a custom component

We can create a Visualforce component to use in a Visualforce page. Navigate to the following path to create a new Visualforce component:

Your Name | Setup | Develop | Components | New

Visualforce Components

Similar to the way functions work in a programming language, custom Visualforce components allow you to encapsulate common patterns in one or more Visualforce pages.

View: [Create New View](#)

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P

Action	Label	Name	Namespace Prefix	Api Version	Description	Created By Alias	Created Date	Last Modified Date
Edit Del	CustomerSummary	CustomerSummary		27.0	CMadu		3/13/2013 10:41 AM	Cl

Create new Visualforce components

We need to specify the following properties while creating a custom component:

- **Label:** This custom component will be identified in the setup tools by using the label.
- **Name:** This custom component will be identified in Visualforce markup by using `name`. This must be unique within the organization.
- **Description:** This gives the description of the custom component.
- **Body:** The Visualforce markup must be placed within the body section.

Component Edit [Save] [Quick Save] [Cancel] [Where is this used?] [Component Reference]

Component Information | = Required Information

Label

Name

Description

Visualforce Markup [Version Settings]

```

1 <apex:component>
2 <!-- Begin Default Content REMOVE THIS -->
3 <h1>Congratulations</h1>
4 This is your new Component
5 <!-- End Default Content REMOVE THIS -->
6 </apex:component>

```



The name of the custom component should begin with a letter, and it should not end with an underscore. Further, spaces or two consecutive underscores should not be included in the name.

The maximum amount of data that a custom component can contain is 1 MB, or approximately 1,000,000 characters.

We can specify the version of Visualforce and the API used with the particular component by using the version setting.

The body of a custom component can be defined as follows:

```
<apex:component
  <!--Desire markup here-->
</apex:component>
```



The component markup is same as other Visualforce pages. It can be a combination of Visualforce and HTML tags. We can also add customized CSS and JavaScript.

All the markup should be defined within the `<apex:component>` tag. Our custom component example is a customer's summary with recent orders. Suppose our custom component name is `customerSummary`, we can use this component in multiple Visualforce pages. The usage is as follows:

```
<apex:page>
  <c: customerSummary />
</apex:page>
```

Custom attributes and custom controllers

When we are creating complex custom components, we need to use some other features to build custom components. Mainly, we have to use custom attributes and custom controllers for custom components. The attributes can be defined in `<apex:component>` for passing values from the Visualforce page (the page that used the component) to the custom component or to the component's controller.

We have implemented the example explained from the beginning of this chapter. The following is the component markup and it contains the attribute's and the component's definitions. We have two attributes to be passed, which are customer ID and the number of recent orders that we want to show in the page. These two parameters are used to pass the values to the component's controller:

```
<apex:component controller="CustomerSummaryComponenetController">
  <!-- Attribute Definitions -->
  <apex:attribute name="customerId" Type="String" required="true"
description="customer id" assignTo="{!CusID}"/>
  <apex:attribute name="noOfRecentOrders" Type="Integer"
required="true" description="Number of recent orders"
assignTo="{!RecentNo}"/>
  <!-- Attribute Definitions : End -->
```

```

<!-- Component Definition -->
<apex:componentBody >
  <apex:pageBlock >
    <apex:pageBlockSection title="Customer Details">
      <apex:outputField value="{!CurrentCustomer.Name}"/>
      <apex:outputField value="{!CurrentCustomer.
Address__c}"/>
      <apex:outputField value="{!CurrentCustomer.
Email__c}"/>
    </apex:pageBlockSection>

    <apex:pageBlockSection title="Recent Order Details">
      <apex:pageBlockTable value="{!RecentOrderList}"
var="order">
        <apex:column value="{!order.Name}"/>
        <apex:column value="{!order.Planned_Delivery_
Date__c}"/>
        <apex:column value="{!order.Status__c}"/>
      </apex:pageBlockTable>
    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:componentBody>
<!-- Component Definition : End -->

</apex:component>

```

The following code snippet shows the custom controller that is associated with the `CustomerSummary` custom component. This controller is used to manipulate the attribute's values. In this example, we have queried the customer record and the recent order details of the particular customer. The query results of `CurrentCustomer` and `RecentOrderList` depend on the `CusID` and `RecentNo` values:

```

public class CustomerSummaryComponenetController{
  public String CusID{get;set;}
  public Integer RecentNo{get;set;}
  public Customer__c CurrentCustomer{
    get{
      CurrentCustomer = new Customer__c();
      CurrentCustomer = [SELECT Id, Name, Address__c, Email__c,
Telephone__c FROM Customer__c WHERE Id =: CusID];
      return CurrentCustomer;
    }
    set;
  }
  public List<Order__c> RecentOrderList{

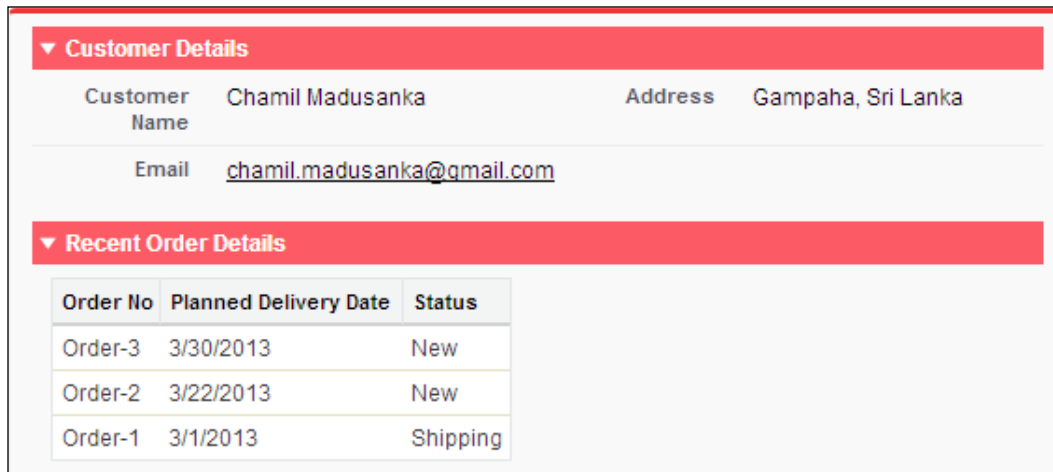
```

```
    get{
        RecentOrderList = new List<Order__c>();
        RecentOrderList = [SELECT Id, Name, Customer__c,
Delivered__c, Planned_Delivery_Date__c, Status__c FROM Order__c WHERE
Customer__c =: CusID ORDER BY CreatedDate DESC LIMIT :RecentNo];
        return RecentOrderList;
    }
    set;
}
}
```

This is the way of using our custom component. Here, we have passed the customer ID and the number of recent orders values that we want to see:

```
<apex:page StandardController="Customer__c">
  <c:CustomerSummary customerId="{!Customer__c.Id}"
noOfRecentOrders="3"></c:CustomerSummary>
</apex:page>
```

The following screenshot shows the result of the `customerSummary` custom component:



Summary

In this chapter, we saw an overview of Visualforce custom components. We now have the knowledge to create and use Visualforce custom components. With the usage of Visualforce custom components, we learned the mechanisms for reusing code in Visualforce. We have seen how to build more customizable custom components by using custom attributes and custom controllers.

5

Dynamic Visualforce Bindings

The dynamic Visualforce binding is one of the greatest features of the Spring '11 release. We can use this feature to build generic Visualforce pages without thinking which record fields have to be shown on the page. The record fields are determined at runtime rather than compile time. This is a powerful feature which allows us to minimize the code (Visualforce and Apex code). Otherwise, we have to write more queries, and have to populate lists of records, and render more fields. Using the dynamic Visualforce binding, we can develop a single page that renders differently for various users based on their authorizations or preferences.

This chapter covers the following topics:

- Using dynamic references with standard objects and custom objects
- Referencing Apex Maps and Lists
- Working with field sets
- Dynamic references to global variables

Let's learn about dynamic Visualforce binding...

Using dynamic references with standard objects and custom objects

Dynamic Visualforce binding is supported for both standard and custom objects in Salesforce. We can use dynamic binding in the following form:

```
reference [expression]
```

Let's discuss the preceding form in detail:

- **reference:** This can be an sObject, an Apex class, or a global variable.
- **expression:** This can be the name of the field or a related object. If it is a related object, then recursively-selected fields or further related objects can be used.



Dynamic bindings can be used in the page where formula expressions are valid. It is used with the `{!}` notation. If it is referenced from an Apex class, then the particular attribute (sObject or variable) must be public or global.



Defining relationships: If there are object relationships to be evaluated in expressions, they become complex expressions. Consider our example, where the `Order__c` custom object has a relationship with the `Customer__c` custom object. The relationship between these two objects is called `Orders__r`. The `Customer__c` object has the `Email__c` field. The same `Email__c` field will be returned by the following dynamically-cast lookups:

- `Order__c.Customer__c['Email__c']`
- `Order__c['Customer__c.Email__c']`
- `Order__c['Customer__c']['Email__c']`
- `Order__c.Orders__r[Email__c]`
- `Order__c[Orders__r.Email__c]`
- `Order__c[Orders__r][Email__c]`

A dynamic Visualforce page must have a standard controller and further implementations can be done in an associated controller extension. The reason is that Visualforce automatically handles the optimization of the SOQL queries performed by the page's `StandardController` or `StandardSetController` object by loading only the used fields.

When we create a page with static references, the page can identify the fields and objects during compilation. Then the `StandardController` object will transform the particular fields and objects into SOQL queries. But the dynamic references are evaluated at runtime and not at compile time. This means that the dynamic references are evaluated after performing the SOQL query of the `StandardController` object. Therefore, when we use dynamic references and we have to provide some extra information to the controller extension, we can use the `addFields()` method to add any number of additional fields. This method will pass a list of additional fields to `StandardController` and those fields will load without giving runtime errors. The usage of the `addField()` method is as follows:

```
public DynamicOrderExtension(ApexPages.StandardController controller)
{
    controller.addFields(editableFields);
}
```

The following example shows the usage of dynamic Visualforce binding. This page shows an order record with some editable fields. Some fields are related to object (`Customer__c`). We can understand the usage of dynamic reference with object relationship traversing.

```
<apex:page standardController="Order__c" extensions="DynamicOrderExtension">
    <apex:pageMessages /><br/>
    <apex:form >
        <apex:pageBlock title="Edit Order" mode="edit">
            <apex:pageBlockSection columns="1">
                <apex:inputField value="{!Order__c.Name}"/>
                <apex:repeat value="{!editableFields}" var="f">
                    <apex:inputField value="{!Order__c[f]}"/>
                </apex:repeat>
            </apex:pageBlockSection>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```


The following code is the controller extension of the preceding Visualforce page. The `DynamicOrderExtension` controller extension has a list of strings called `editableFields` and this string list contains some fieldnames of the `Order__c` object and some fields of related object (`Customer__c`) of `Order__c`. In this example, editable fields are hardcoded. But we can get information for your dynamic references by using the Apex's `Schema.sObjectType` methods. This will make a more dynamic and powerful reference. For example, `Schema.SObjectType.Order__c.fields.getMap()` returns a map with the name of the `Order__c` fields. The preceding markup has the `<apex:repeat>` tag, which is used to loop the `editableFields` string list and the `<apex:inputField>` tag which displays that particular returned string. It represents the field names of the order and the related object's field names. The following markup line displays the dynamic reference:

```
<apex:inputField value="{!Order__c[f]}" />

public with sharing class DynamicOrderExtension {
    public final Order__c orderDetails { get; private set; }

    public DynamicOrderExtension(ApexPages.StandardController
controller) {
        String qid = ApexPages.currentPage().getParameters().
get('id');
        String theQuery = 'SELECT Id, ' + joinList(editableFields, ',
') +
        ' FROM Order__c WHERE Id = :qid';
        this.orderDetails = Database.query(theQuery);
        controller.addFields(editableFields);
    }

    public List<String> editableFields {
        get {
            if (editableFields == null) {
                editableFields = new List<String>();
                editableFields.add('Delivered__c');
                editableFields.add('Customer__c');
                editableFields.add('Planned_Delivery_Date__c');
                editableFields.add('Status__c');
                editableFields.add('Customer__r.Email__c');
            }
            return editableFields ;
        }
        private set;
    }

    private static String joinList(List<String> theList, String
separator) {

        if (theList == null) {
```

```

        return null;
    }

    if (separator == null) {
        separator = '';
    }

    String joined = '';
    Boolean firstItem = true;

    for (String item : theList) {
        if (null != item) {
            if (firstItem) {
                firstItem = false;
            }
            else {
                joined += separator;
            }
            joined += item;
        }
    }
    return joined;
}
}

```

This page needs to be accessed with the ID of a valid case record specified as the id query parameter. For example, <https://c.ap1.visual.force.com/apex/DynamicBindingExample?id=a029000000086Hlr>.

Referencing Apex Maps and Lists

Apex Maps and Lists can be dynamically referred in a Visualforce page. Apex Lists are vastly used with the `<apex:pageBlockTable>` and `<apex:repeat>` tags. In our preceding example (under the *Using Dynamic references with Standard objects and custom object* section) we have already seen the dynamic references of Apex List. The following example shows the dynamic reference of an Apex Map. This is the markup of the `DynamicExampleListMap` page:

```

<apex:page controller="DynamicBindingsMapListExample">
  <apex:form >
    <apex:actionFunction name="reDisplayCustomers" rerender="cust" />
    <apex:pageBlock title="Criteria">
      <apex:outputLabel value="Starting Letter"/>
      <apex:selectList value="{!selectedKey}" size="1" onchange="reDi
splayCustomers()" >
        <apex:selectOptions value="{!keys}" />
      </apex:selectList>
    </apex:pageBlock>
  </apex:form>
</apex:page>

```

```
</apex:pageBlock>
<apex:pageBlock title="Customers">

    <apex:outputPanel id="cust">
        <apex:pageBlockTable value="{!customerMap[selectedKey
    ]}" var="cus">
            <apex:column value="{!cus.name}"/>
            <apex:column value="{!cus.Address__c}"/>
            <apex:column value="{!cus.Email__c}"/>

        </apex:pageBlockTable>
    </apex:outputPanel>
</apex:pageBlock>
</apex:form>
</apex:page>
```

The following is the related custom controller. The `customerMap` object contains the customer's records and the pick list is dynamically filled with the appropriate values from the Map. We can select a letter from the pick list and the customer list, and then rearrange the result according to the selected letter. The `customerMap` object returns the corresponding customer list at runtime by using the `{!customerMap[selectedKey]}` dynamic reference:

```
public class DynamicBindingsMapListExample
{
    public Map<string, List<Customer__c>> customerMap {get; set;}
    public List<selectoption> keys {get; set;}
    public String selectedKey {get; set;}
    public Map<string, Customer__c> custByName {get; set;}

    public Set<string> getMapKeys()
    {
        return customerMap.keySet();
    }

    public DynamicBindingsMapListExample()
    {
        custByName = new Map<string, Customer__c>();
        List<string> sortedKeys=new List<string>();
        customerMap = new Map<string, list<Customer__c>>();
    }
}
```

```
customerMap.put('All', new List<Customer__c>());
List<Customer__c> customers = [SELECT Id, Name, Email__c,
Address__c FROM Customer__c ORDER BY Name asc];

for (Customer__c tempCustomer : customers)
{
    customerMap.get('All').add(tempCustomer);
    String start = tempCustomer.Name.substring(0,1);
    List<Customer__c> custFromMap = customerMap.get(start);
    if (custFromMap == null)
    {
        custFromMap =new List<Customer__c>();
        customerMap.put(start, custFromMap);
    }
    custFromMap.add(tempCustomer);
    custByName.put(tempCustomer.name, tempCustomer);
}

keys=new List<selectoption>();
for (String key : customerMap.keySet())
{
    if (key != 'All')
    {
        sortedKeys.add(key);
    }
}
sortedKeys.sort();
sortedKeys.add(0, 'All');

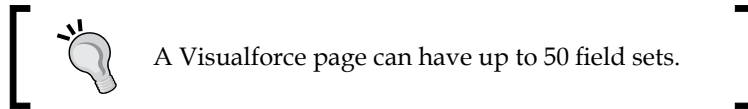
for (String key : sortedKeys) {
    keys.add(new SelectOption(key, key));
}
selectedKey='All';
}
}
```

Working with field sets

A field set is a group of fields which can be defined in a declarative manner. Field sets are available in Visualforce pages in the API Version 21.0. These field sets can be displayed on a Visualforce page by dynamic binding. For example, suppose we have created a field set (field set name: `CustomerDetails`) with the `Email__c`, `Name`, and `Address__c` fields of the customer object. We can refer to the `CustomerDetails` field set in Visualforce as follows:

```
<apex:page standardController="Customer__c">
  <apex:repeat value="{!$ObjectType.Customer__c.FieldSets.
CustomerDetails}" var="f">
    <apex:outputText value="{! Customer__c [f]}" /><br/>
  </apex:repeat>
</apex:page>
```

When we want to create a managed package or add/ remove/reorder fields in the field set, we can accomplish that without modifying any code.



Summary

In this chapter, we learned about the powerful feature of dynamic binding, which was released by Spring'11. We became familiar with the usage of standard and custom object dynamic references. And we acquired a good knowledge of referencing Apex Maps/List and the way of using field sets. We have also seen the usage of dynamic reference of global variables. With all these, we learned the mechanisms of minimizing the Visualforce and Apex code.

6

Visualforce Charting

Visualforce charting is one of the best features from the Winter '13 release. It is a collection of components which provides a simple way to create charts on our Visualforce pages and Visualforce custom components. This feature gives us the facility to customize charts that are based on our data sets from SOQL queries, and create custom charts (such as pie, bar, and line charts) on our Visualforce pages. We can create charts with Visualforce and Apex, and the charting component takes care of all of the JavaScript code for us. Visualforce charts are rendered by using JavaScript on the client side and it allows us to build animated and visually excited charts on the Visualforce pages.

Sometimes, the standard Salesforce charts and dashboards may be insufficient to meet our requirements. This is where Visualforce charting comes into play. When we cannot fulfill our requirement with Visualforce charting we can use Google charts in Visualforce.

This chapter explains Visualforce charting, which is a collection of components that provides a simple and intuitive way to create charts in your Visualforce pages and custom components. The following topics will be covered in this chapter:

- Limitations and considerations of Visualforce charting
- How does Visualforce charting work
- A complex chart with Visualforce charting

Let's build some exciting Visualforce charts...

Limitations and considerations of Visualforce charting

When Force.com released the Visualforce charting feature, they announced a few known limitations and considerations for Visualforce charting, which are as follows:

- Visualforce charting can be rendered only in **Scalable Vector Graphics (SVG)** supported browsers.
- Visualforce charts cannot be displayed in pages rendered as PDFs because Visualforce charting uses JavaScript to draw the charts.
- Visualforce charting is not used in e-mail messages or e-mail templates because e-mail clients do not support JavaScript's execution in e-mail messages.
- When we develop a Visualforce page with Visualforce charting, we need to use a JavaScript debugging tool such as Firebug to track errors and messages returning from Visualforce charting to the JavaScript console.
- Dynamic Visualforce charting (Apex-generated) is still (as of Spring 2013) not supported by the Force.com platform. However, this feature is supposed to be released soon.

How does Visualforce charting work

Visualforce charting relies on Apex, Visualforce, and JavaScript. When we create Visualforce charts, we need an Apex method to prepare or query data to use as the source of the chart. Then we need to define our chart by using Visualforce charting components. The chart data that is prepared in the Apex method is bound to the chart component and the JavaScript draws the chart in the browser.

A Visualforce chart needs a chart container that has at least one data series component. We have the ability to add additional series, chart axes, and labeling components (such as legend, chart labels, and tool tips for data points).

The following example creates a simple pie chart that contains the number of items that are delivered to customers. In this example, we have hardcoded values for the chart data source. The following is the markup of a pie chart example:

```
<apex:page controller="VFChartController" title="Pie Chart">
  <apex:chart height="350" width="450" data="{!chartData}">
    <apex:pieSeries dataField="data" labelField="name"/>
    <apex:legend position="right"/>
  </apex:chart>
</apex:page>
```

Here is the associated custom controller, which prepares the data source for the chart. The chart container is defined by the `<apex:chart>` component and data binding is done by the `getChartData()` controller method. The `<apex:pieSeries>` component defines the label and data field from the returned data as follows:

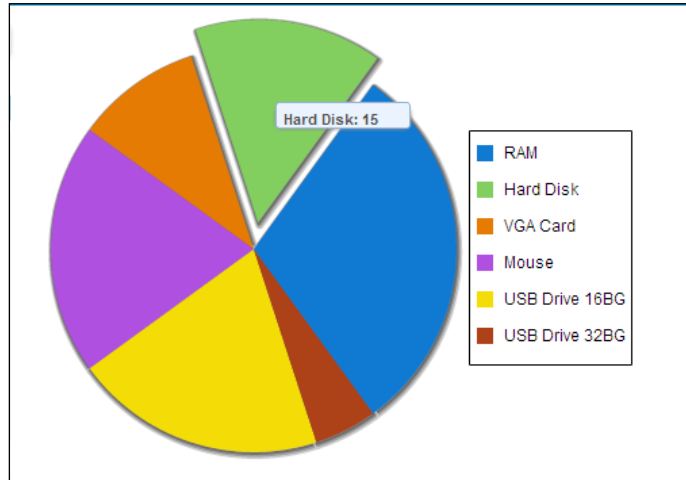
```
public class VFChartController {

    public List<PieChartData> getChartData () {
        List<PieChartData> data = new List<PieChartData>();
        data.add(new PieChartData('RAM', 30));

        data.add(new PieChartData('Hard Disk', 15));
        data.add(new PieChartData('VGA Card', 10));
        data.add(new PieChartData('Mouse', 20));
        data.add(new PieChartData('USB Drive 16BG', 20));
        data.add(new PieChartData('USB Drive 32BG', 5));
        return data;
    }
    // Wrapper class
    public class PieChartData {
        public String name { get; set; }
        public Integer data { get; set; }

        public PieChartData(String name, Integer data) {
            this.name = name;
            this.data = data;
        }
    }
}
```


The resultant chart of the preceding example is shown in the following screenshot:



The resultant pie chart

The preceding example illustrates the following points:

- `PieChartData`: This is an inner class which has a set of properties to define the label (the `name` property) and the data (the `data` property) of the chart.
- `getChartData()`: This method returns a list of wrapper objects of `PieChartData`. These list elements create the data points for the chart.
- `<apex:pieSeries>`: This component defines the label and data field from the returned data (objects of `PieChartData`).

Providing chart data

The following are the three different ways to provide data source to the chart:

- Using the controller method
- Using a JavaScript function
- Using a JavaScript array

Using the controller method

This technique has been illustrated in our simple pie chart example. This is a server-side technique and here we have used a controller method to return a list of objects. This object list can be our own Apex wrapper objects (as in our previous example), `AggregateResult` objects, or `sObjects`. The result of the method is serialized to JSON on the server side, and the result is directly used by the `<apex:chart>` component on the client side. Refer to the simple pie chart example for this technique.

Using a JavaScript function

There is another way to provide data to the chart component that is via a JavaScript function. We can use the name of the JavaScript function in the `<apex:chart>` component. This JavaScript function is the data provider and it can be defined in the Visualforce page or linked to the Visualforce page. We can use this JavaScript function to manipulate the data before sending it to the `<apex:chart>` component. See the JavaScript Remoting for Apex Controllers section for more information about using JavaScript remoting in Visualforce. The following is a simple example of a JavaScript function with the `<apex:chart>` component:

```
<apex:page controller="VFRemoteChartController" title="Pie Chart">
  <script>
    function getRemoteChartData(callback) {
      VFRemoteChartController.getRemotePieChartData(function(result,
event) {
        if(event.status && result && result.constructor === Array) {
          callback(result);
        }
      });
    }
  </script>
  <apex:chart height="350" width="450" data="getRemoteChartData">
    <apex:pieSeries dataField="data" labelField="name"/>
    <apex:legend position="right"/>
  </apex:chart>
</apex:page>
```

The following class is the associated custom controller of the preceding page. We have defined the remote method with the `@RemoteAction` annotation. That remote method transforms the data into the chart component:

```
public class VFRemoteChartController {

    @RemoteAction
    public static List<PieChartData> getRemotePieChartData() {
        List<PieChartData> data = new List<PieChartData>();
        data.add(new PieChartData('RAM', 30));

        data.add(new PieChartData('Hard Disk', 15));
        data.add(new PieChartData('VGA Card', 10));
        data.add(new PieChartData('Mouse', 20));
        data.add(new PieChartData('USB Drive 16BG', 20));
        data.add(new PieChartData('USB Drive 32BG', 5));
        return data;
    }

    // Wrapper class
    public class PieChartData {
        public String name { get; set; }
        public Integer data { get; set; }

        public PieChartData(String name, Integer data) {
            this.name = name;
            this.data = data;
        }
    }
}
```

Using a JavaScript array

Another way of providing data is by using a JavaScript array. We can use Visualforce charting without using any custom controller by using a JavaScript array which can be Salesforce data or non-Salesforce data. We can query the Salesforce data in JavaScript code by using the Ajax Toolkit API which is a JavaScript wrapper around the API and we can build non-Salesforce data sources by using a JavaScript array in our own JavaScript code. Then we can use the array in the chart component by providing the name of the array to the `<apex:chart>` component. This method is useful when your data source relies only on the client side, and not on the server side. The following example illustrates how to define a Visualforce chart with a JavaScript array:

```

<apex:page >
  <script>
    // Build the chart data array in JavaScript
    var dataArray = new Array();
    dataArray.push({'data':15,'name':'Hard Disk'});
    dataArray.push({'data':10,'name':'VGA Card'});
    dataArray.push({'data':20,'name':'Mouse'});
    dataArray.push({'data':20,'name':'USB Drive 16BG'});
    dataArray.push({'data':5,'name':'USB Drive 32BG'});
  </script>
  <apex:chart height="350" width="450" data="dataArray">
    <apex:pieSeries dataField="data" labelField="name"/>
    <apex:legend position="right"/>
  </apex:chart>
</apex:page>

```



Visualforce charts are more customizable. We can customize the look and feel of elements, markers, the opacity of fill colors/lines, and can combine various data sources.

A complex chart with Visualforce charting

We can use Visualforce charting to build complex charts that represent various data series in one chart. For example, we can build a chart with multiple data series. The following example shows the number of items sold in three different years. The code shows the custom controller the custom controller. The `getComplexChartData()` method prepares the data for the chart component. This controller also has a `@RemoteAction` method to get the data to the chart component. But this example hasn't used the JavaScript remoting. It illustrates the way of reusing the data generation method for both server-side and JavaScript remoting methods.

```

public class ComplexChartController{

    // Return a list of data points for a chart
    public List<ChartData> getVFChartData() {
        return ComplexChartController.getComplexChartData();
    }

    // Make the chart data available via JavaScript remoting
    @RemoteAction

```

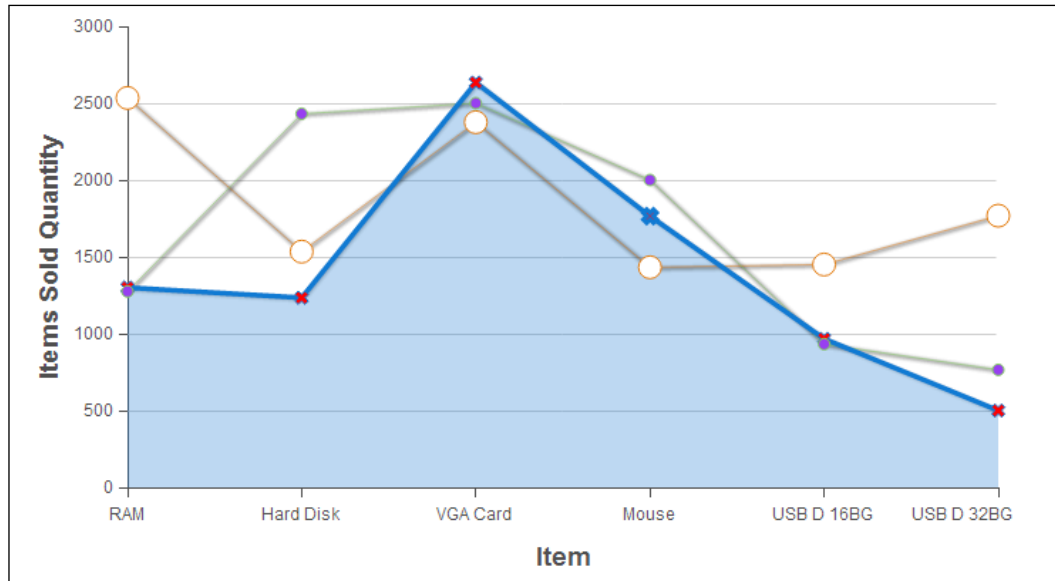
```
public static List<ChartData> getRemoteVFChartData() {
    return ComplexChartController.getComplexChartData();
}

//prepare data sources
public static List<ChartData> getComplexChartData() {
    List<ChartData> data = new List<ChartData>();
    data.add(new ChartData('RAM', 1300, 1275, 2534));
    data.add(new ChartData('Hard Disk', 1234, 2431, 1534));
    data.add(new ChartData('VGA Card', 2634, 2500, 2376));
    data.add(new ChartData('Mouse', 1765, 2000, 1432));
    data.add(new ChartData('USB D 16BG', 967, 932, 1450));
    data.add(new ChartData('USB D 32BG', 500, 765, 1768));
    return data;
}

// Wrapper class
public class ChartData{
    public String name { get; set; }
    public Integer data1 { get; set; }
    public Integer data2 { get; set; }
    public Integer data3 { get; set; }

    public ChartData(String name, Integer data1, Integer data2,
Integer data3) {
        this.name = name;
        this.data1 = data1;
        this.data2 = data2;
        this.data3 = data3;
    }
}
}
```

The following screenshot represents a chart with three line series. This chart illustrates the number of items sold in three different years:



The output of the complex chart example



The x and y axes need to be defined in line and bar charts.



The markup of the complex chart example is as follows:

```
<apex:page controller="ComplexChartController">
  <apex:chart height="400" width="700" data="{!vFChartData}">

    <apex:axis type="Numeric" position="left" fields="data1"
    title="Items Sold Quantity" grid="true"/>
    <apex:axis type="Category" position="bottom" fields="name"
    title="Item"> </apex:axis>
  </apex:chart>
</apex:page>
```

```
<apex:lineSeries axis="left" fill="true" xField="name"
yField="data1" markerType="cross" markerSize="4"
markerFill="#FF0000"/>
  <apex:lineSeries axis="left" xField="name" yField="data2"
markerType="circle" markerSize="4" markerFill="#8E35EF"/>
  <apex:lineSeries axis="left" xField="name" yField="data3"
markerType="circle" markerSize="8" markerFill="#FFFFFF"/>
</apex:chart>
</apex:page>
```

Summary

In this chapter, we became familiar with Visualforce charting which allows us to build customized charts based on our data. We learned that Visualforce charting is a JavaScript-based feature. Therefore, we have learned about the limitations and considerations of Visualforce charting. We have also seen how to create simple and complex charts by using Visualforce charting components.

7

Visualforce for Mobile

Using Visualforce and Apex we can build complex, dynamic, and powerful native applications on the Force.com platform. Nowadays, users are not satisfied only with only a web application, due to the competition in the industry. When it comes to applications on the Force.com platform, we can extend them to mobile devices by using Visualforce Mobile. Visualforce Mobile is a hybrid of client-side and on-demand programming. This allows us to extend applications for mobile devices with offline data access flexibility. We can rapidly build apps for mobile devices by using Visualforce and Apex.

This chapter covers how to extend applications built on the Force.com platform for mobile devices, and how developers can use Visualforce Mobile. The following topics will be covered in this chapter:

- Understanding Salesforce Mobile
- Developing and mobilizing Visualforce pages

Let's build Visualforce for mobiles.

Understanding Salesforce Mobile

Salesforce Mobile is a client application provided by Salesforce. Salesforce Mobile is used to extend applications which are built on the Force.com platform. This allows us to access the Salesforce data from BlackBerry, iPhone, or Windows Mobile devices. The Salesforce Mobile client application has the following features:

- Makes the interaction with Salesforce via wireless carrier networks
- Stores a local copy of user's data on the mobile device. It uses its own database.

A set of parameters which is called as **Mobile configuration** is determined by the data sent to the mobile device. It defines the relevant subset of the user's Salesforce records.



If we access the Salesforce data via a mobile device, we need a separate Salesforce Mobile user license for a particular user. The Developer and Ultimate editions have only one mobile license. Others must be purchased separately.

Salesforce Mobile and Visualforce Mobile supporting devices

Salesforce Mobile is supported on BlackBerry, iPhone, and Windows Mobile. Currently, Windows Mobile is not supported for Visualforce Mobile. BlackBerry and iPhone devices must have following requirements:

- **BlackBerry configurations are as follows:**
 - Supported OS versions 4.3 through 7.0 and for better performance versions 4.6 through 4.7
 - A minimum of 5 MB of free memory should be available on the device
 - The mobile client application is supported on BlackBerry 8100 Series (Pearl), BlackBerry 8300 Series (Curve), BlackBerry 8800 Series, BlackBerry 8900 Series (Javelin), BlackBerry 9000 Series (Bold), and BlackBerry 9500 Series (Storm)



You don't need to own an iPhone or a BlackBerry device to develop and test applications. You can use simulators.

- **iPhone configurations are as follows:**
 - Supported for latest iPhone OS
 - A minimum of 5 MB of free memory should be available on the device
 - The mobile client application is supported on iPhone, iPhone 3G, iPhone 3GS, and iPod Touch

Capabilities and limitations of the mobile application

The native client application of Salesforce Mobile has an embedded browser which is used to communicate between a client application and a Visualforce page. There are a few concerns to consider when we are using Salesforce Mobile, which are as follows:

- Accounts, assets, contacts, opportunities, tasks, leads, events, price books, products, cases solutions, and custom objects can be mobilized.
- Custom links, s-controls, mashups, merge fields, and image fields cannot be mobilized.
- Workflow rules, validation rules, formula fields, and Apex triggers are not suitable to be run on the mobile device. However, they can be run on the server side after a record is saved and submitted to Salesforce.
- User permissions, record types, and page layouts are inherited from Salesforce. However, the administrator can change them to restrict the permissions of mobile users.
- When we add a child data set to a parent data set, the object becomes a related list on the mobile device.
- Reports are available only in a BlackBerry client application, but dashboards are available in both iPhone and BlackBerry.
- Sorting, summaries, subtotals, or grouping are not supported in the report viewer of the mobile application.
- Custom views of Salesforce can be accessed by iPhone and BlackBerry users, but the custom views can be created only by BlackBerry users.
- In the mobile application, custom views are limited to two columns.
- Mobilized Visualforce tabs and web tabs can be accessed in the client application by both iPhone and BlackBerry users.



The embedded browser communicates with Salesforce using the device's internet connection; the native client application communicates with Salesforce asynchronously through the SOAP API. The embedded browser can execute JavaScript, but the native client application cannot.

Using Visualforce Mobile

When we use mobile applications, they are client-side applications and they need an installation. The mobile applications need a periodic connection to send and receive data. When compared with the mobile on-demand applications, the mobile client applications are dependent upon network connection and speed. Mobile client applications can be used to work offline as well. When we come to speed, wireless data networks are still very slow. But client applications are highly responsive.

There are situations where a native client application cannot satisfy the customer's needs. Therefore, we can use Visualforce Mobile in the following situations:

- Integrating with other web APIs, for example, Google Maps
- Rebuilding the functionality which are not available in client applications, such as responding to an approval request
- Mobilizing a standard Salesforce object which is not supported in a client application
- Integrating with peripheral devices, such as Bluetooth or embedded GPS
- Overriding the action of the standard buttons on the detail page

Developing and mobilizing Visualforce pages

Developing a Visualforce page for a Salesforce Mobile is different from developing pages for Salesforce, especially as the user experiences on desktop browsers and mobile browsers are different.

Best practices for building Visualforce Mobile pages for iPhone and BlackBerry

The following are the best practices:


- **Controllers:** Salesforce Mobile supports custom objects and many standard objects. Standard layouts and styles of a standard page are too complex for a mobile browser. When developing pages for the mobile application, the best practice is to use custom controllers for the page. If your controller has a very complex business logic, it may slow down the loading of the page.

- **Header and sidebar:** Remove the header and the sidebar from the Visualforce Mobile pages because they may lead to long loading times and there won't be sufficient space to display them on the mobile screen. They can be removed by using the following code:

```
<apex:page showHeader="false" sidebar="false">
```


- **Page styles:** The standard stylesheets of Salesforce pages are too complex for the mobile browser. We have to stop loading the standard stylesheets by using the `standardStylesheets` attribute of the `<apex:page>` tag as follows:

```
<apex:page showHeader="false" sidebar="false"
standardStylesheets="false">
  <style type="text/css">
<!--your custom styles here-->
  </style>
</apex:page>
```

 The best approach to add a stylesheet to your page is to include a `<style>` section just below the `<apex:page>` tag.

- **Reuse:** In a mobile client application, reusing is a key component. We can create a Visualforce page with custom styles and reuse that page in other Visualforce pages using the `<apex:include>` component. For example, if the preceding page's name is `myStylePage` and we have implemented a custom style on that page, then we can include the preceding page with styles as follows:

```
<apex:page standardStylesheets="false"/>
<apex:include pageName="myStylePage"/>
</apex:page>
```

 We can create mobile-optimized stylesheet as a static resource and we can refer to the same stylesheet in non-mobile pages. We can increase the page's loading time by using the stylesheet as a static resource.

- **Lookups:** Lookups doesn't work properly on BlackBerry and iPhone. Therefore the best practice is to validate the entry in the lookup upon saving the record. We can use an Apex trigger for validation. And we can also change the file type occasionally.

iPhone considerations

While developing pages for the iPhone, we must consider following things:

- **Page zoom:** To maximize the compatibility with a broad range of websites, the iPhone browser sets the page width to 980 pixels. Using the `<meta>` tag, the iPhone browser can identify the width to display the page. The following tag definition is only valid for the iPhone browser, other browsers ignore this tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
```
- **URL targets:** The embedded browser doesn't support the `target="_blank"` attribute. The page will not load with this attribute.
- **Screen rotation:** Rotating the screen will not cause the page to flip and resize.
- **Static resource caching:** The embedded browser doesn't support caching.
- **File access:** The embedded browser does not natively offer access to the filesystem, camera, location, or other device data.



General rule for mobile development


We shouldn't use components with JavaScript to perform an action on components that depend on Salesforce stylesheets. If we can see the `<script>` tag that refers to a JavaScript (.js) file or a `<link>` tag that refers to a stylesheet (.css) in the HTML source of our page, the page can fall under the preceding category.

BlackBerry considerations

When developing pages for BlackBerry smartphones, the following considerations are applied:

- **JavaScript Support:** Inline DOM events doesn't work in the BlackBerry browser. The BlackBerry browser has limited JavaScript support. When developing Visualforce pages for BlackBerry, avoiding JavaScript is the best option.

- **Forms and view state:** If you want to use the `<apex:form>` tag in your Visualforce Mobile page, use standard HTML forms instead of `<apex:form>`. If we use the `<apex:form>` tag, the view state of the page will be too large for the BlackBerry browser. When we use a standard HTML tag, we have to do the following things manually and we cannot use the `<apex:commandLink>` tag and `<apex:commandButton>` components:
 - Maintaining state between pages
 - Redirecting to another page

 Parameters sent from the form can be retrieved using the `ApexPages.currentPage().getParameters()` map in the controller.

- **Page styles:** We have to follow the best practices for building Visualforce Mobile pages for iPhone and BlackBerry. Additionally, we must know that the BlackBerry browser ignores some CSS properties, for example, `margin-left`.
- **Line breaks:** The `
` tag is ignored unless there is something on the line.
- **Navigation:** There isn't any in-built navigation in the embedded browser of a BlackBerry client application. We have to provide the navigation links.

Developing cross-platform compatible pages

When we build Visualforce Mobile pages to perform well on both iPhone and BlackBerry browsers, we need to follow the following approaches which are provided by Salesforce:

- **Separation and redirection:** We can build the Visualforce Mobile pages to redirect to a suitable optimized page (iPhone-optimized or BlackBerry-optimized) by using JavaScript. For that, we have to build pages separately for iPhone and BlackBerry. When the connecting device is not a BlackBerry device, the page will redirect the page to an iPhone-optimized page, as given in the following code:

```
<apex:page>
<language="javascript" type="text/javascript">
if(!window.blackberry) {
window.location.href='{!$Page.iPhoneOptimizedVersion}';
}
</script>
</apex:page>
```

- **Conditional code:** The server identifies the connecting device (iPhone or BlackBerry) by using the user agent string in the header, which is the user agent string that is passed by the browser to the server. Therefore, we can build device-conditional code and styles for well-performing pages on both iPhone and BlackBerry devices. The advantage is that the markup is interpreted on the server side and the user gets only the suitable markup which is selected by the conditional code. The disadvantage is that the code can be more complex due to the conditional code. The following example shows the way of handling conditional code and the markup. The markup has two `<apex:outputPanel>` components, each of which renders the markup for a particular device:

```
<apex:page controller="ConditionalCodeController">
  <apex:outputPanel rendered="{!deviceType = 'BlackBerry'}">
    <apex:outputText value="This is BlackBerry"></apex:outputText>
  </apex:outputPanel>

  <apex:outputPanel rendered="{!deviceType = 'iPhone'}">
    <apex:outputText value="This is iPhone"></apex:outputText>
  </apex:outputPanel>
</apex:page>
```

The controller of the preceding markup is as follows and it evaluates the user agent and prepares the `deviceType` attribute in order to render a correct output panel:

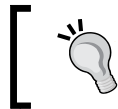
```
public class ConditionalCodeController {
    public String deviceType { get; set; }

    public ConditionalCodeController() {
        String userAgent = ApexPages.currentPage().getHeaders().
get('USER-AGENT');
        if (userAgent.contains('iPhone')) {
            deviceType = 'iPhone';
        }
        else if (userAgent.contains('BlackBerry')) {
            deviceType = 'BlackBerry';
        }
    }
}
```

- **Lowest common denominator:** Build to the lowest common denominator and include only minimal, unobtrusive JavaScript, avoiding scripts with inline events in the tags. Depending on the devices in the customer's organization, you might need to avoid JavaScript all together. On older BlackBerry smartphones, using any JavaScript code can malfunction the page.

Using the JavaScript library

The JavaScript library contains commands for trigger actions in Salesforce Mobile. This JavaScript library can be used to build seamless user experience between a native client application and Visualforce Mobile pages.



The JavaScript commands work only on JavaScript-enabled devices.

The following are the functions in the JavaScript library:

- `mobileforce.device.sync()`: This function forces the mobile client application to synchronize with Salesforce, which updates data records on the device.
- `mobileforce.device.close()`: This function closes the embedded browser containing the Visualforce page and returns the user to the original/previous tab or record.
- `mobileforce.device.syncClose()`: This function forces the mobile client application to synchronize with Salesforce and closes the embedded browser containing the Visualforce page.
- `mobileforce.device.getLocation()`: This function obtains the GPS coordinates of the device's current location.

The following example has the usage of all the commands available in the JavaScript library:

```
<apex:page showheader="false">
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title>Visualforce Mobile Trigger Test</title>
      <!-- <meta name="viewport" content="width=device-width;
initial-scale=1.0; maximum-scale=1.0; user-scalable=0;" />-->

      <!-- Using static resource -->
      <script type="application/x-javascript" src="/mobileclient/
api/mobileforce.js">
      </script>
      <script>
        function sync() {
          mobileforce.device.sync();
          return false;
        }
        function doClose() {
```



```
        mobileforce.device.close();
        return false;
    }
    function syncClose() {
        mobileforce.device.syncClose();
        return false;
    }
    updateLocation = function(lat,lon) {
        document.getElementById('lat').value = lat;
        document.getElementById('lon').value = lon;
    }
    function getLocation(){
        mobileforce.device.getLocation(updateLocation);
        return false;
    }
</script>
</head>
<body>
    <h2>Triggers:</h2>
    <p>
        <a href="#" onclick="return sync();">JS sync</a><br/>
        <a href="#" onclick="return doClose();">JS close</a><br/>
        <a href="#" onclick="return syncClose();">JS sync and
close</a><br/>
        <a href="mobileforce:///sync">HTML sync</a><br/>
        <a href="mobileforce:///close">HTML close</a><br/>
        <a href="mobileforce:///sync/close">HTML sync and close</
a><br/>
    </p>
    <h2>Location:</h2>
    <p>Latitude: <input type="text" disabled="disabled" id="lat"
name="lat" value=""/></p>
    <p>Logitude: <input type="text" disabled="disabled" id="lon"
name="lon" value=""/></p>
    <a href="#" onclick="return getLocation();">Get location</
a><br/>
</body>
</html>
</apex:page>
```

Building a mobile-ready Visualforce tab

After building the Visualforce Mobile pages, we have to do some configuration to access these pages via Salesforce Mobile. To mobilize the Visualforce page, we can create web and Visualforce tabs using the **Mobile Ready** attribute. After navigating to the following path and clicking on **New** in the Visualforce tab we will be directed to a page where we will create a mobile-ready Visualforce tab:

YourName | Setup | Create | Tab

New Visualforce Tab Help for this Page

Step 1. Enter the Details Step 1 of 3

Choose the page for this new tab. Fill in other details.

Select an existing page or [create a new page now](#).

Visualforce Page: VFMobileJSLibrary [VFMobileJSLibrary]

Tab Label: Mobile JS

Tab Name: Mobile_JS i

Tab Style: Bridge 🔍

Mobile Ready: [What Is This?](#)

(Optional) Choose a Home Page Custom Link to show as a splash page the first time your users click on this tab.

Splash Page Custom Link: --None--

Building a mobile-ready Visualforce tab

The **Mobile Ready** checkbox is used to specify whether the Visualforce page displays and functions properly in a mobile browser. By selecting the **Mobile Ready** checkbox, we can add the tab to the list of available tabs for our mobile configuration.

Creating the mobile configuration

When we use Salesforce Mobile, we have to determine the mobile user's data and the permissions given to mobile users. Mobile configuration is a set of parameters. An organization can have multiple mobile configurations for different kinds of mobile users. When we create a mobile configuration, a particular user must be assigned to a mobile license.



If the user account's **Mobile User** checkbox is checked, then this means that the user is already identified as a mobile user, unless we give the **Manage Mobile Configurations** permission to the user's profile or the permission set.

To create a mobile configuration, navigate to:

YourName | Setup | Mobile Administration | Salesforce Mobile | Configuration | New Mobile Configuration

Only the active mobile configuration will be available for use. Optionally, we can select the **Mobilize Recent Items** option to mark recently the used records in Salesforce for device synchronization and the **Mobilize Recent Items** option to select a value from the **Maximum Number of Recent Items** drop-down list. In the mobile configuration, we can add an individual user or we can use a profile to give the permissions. We don't sync if the data size exceeds property to avoid overloading of mobile device. To do that, we have to specify the maximum size of data that is allowed for all data sets combined in this mobile configuration.



Salesforce.com will not synchronize any data sets if the combined data size exceeds this limit.

After creating the mobile configuration, we can define the data set by adding objects and records, and automatically synchronizing the mobile devices.

After developing Visualforce Mobile pages, we have to test them for checking the functionality and the appearance. For that, we can install the mobile application on a BlackBerry or an iPhone device or we can use the iPhone or BlackBerry simulator.

Summary

In this chapter, we learned about Salesforce Mobile and the usage of Salesforce Mobile. We became familiar with the way of extending applications built on the Force.com platform to mobile devices and how we can use Visualforce Mobile. We understood how to develop and mobilize Visualforce pages. We have seen the supporting configurations of mobile devices for a Visualforce Mobile.

8

Best Practices for Visualforce Developments

Visualforce pages are the replacement for Salesforce standard pages. When we use Visualforce pages, the delay experiences and unexpected behaviors must not be there. Therefore, we have to follow the best practices to improve the user experience and coding standards during the Visualforce developments, in order to improve the user experience. There are some situations and components where we can apply some best practices. This chapter will cover the following topics:

- Accessing component IDs
- Page block components
- Controllers and controller extensions
- Improving Visualforce's performance
- Static resources
- Rendering PDFs
- Using component facets

Let's build Visualforce pages with super performance...

Accessing component IDs

When we refer Visualforce components in JavaScript, the ID attribute plays a major role. Every Visualforce component has an ID attribute. The ID attribute must be specified to a particular component in order to refer to it in JavaScript and it is used to bind the two components together. When the page is rendered, this ID attribute is a part of DOM ID of the particular component. The ID attribute must be unique as well. The following best practices are applied for accessing component IDs:

- Use the `$.Component` global variable to simplify access. For an example, when we have an input field with `id="inputOne"` within a page block with `id="blockOne"`, we can access the input field with the `$.Component.blockOne.inputOne` expression.
- No need to specify an ID for a component you want to access if it is an ancestor or sibling to the `$.Component` variable in the Visualforce component's hierarchy.

Page block components

The `<apex:pageBlockSectionItem>` component can have only two child components. With the customer requirements and the developing requirements, there can be more than two child elements inside `<apex:pageBlockSectionItem>`. Using `<apex:outputPanel>` we can add more than two elements in `<apex:pageBlockSectionItem>` as follows:

```
<apex:pageBlock>
  <apex:pageBlockSection>
    <apex:pageBlockSectionItem>
      <apex:outputLabel value="LabelName"/>
      <apex:outputPanel>
<!--We can add our extra child components here within the
apex:outputPanel -->
        </apex:outputPanel>
      </apex:pageBlockSectionItem>
    </apex:pageBlockSection>
  </apex:pageBlock>
```

Controllers and controller extensions

When we are developing controllers and controller extensions that are associated to Visualforce pages, we need to adhere to the following best practices:

- By using the `with sharing` keyword, we can enforce the sharing rules in controllers. Then the code will execute in the user mode instead of the system mode.
- We must not depend on the setter method to be executed before the constructor.
- We must not depend on the execution order or side effects while creating custom methods in a custom controller or a controller extension.
- Do not use DML operations inside a loop.
- While performing record filtering, add filters in the following order:
 - In SOQL
 - In Apex
 - In Visualforce
- If possible, calculations must be performed in SOQL instead of Apex.

Improving Visualforce's performance

The performance of a Visualforce page is a key factor to consider in development because performance is a reason that affects the end user's satisfaction of the application. The following are the best practices to improve Visualforce's performance:

- Use only one `<apex:form>` tag per Visualforce page because each `<apex:form>` tag adds a view state to the page. A Visualforce page has a limit for view state size that is 135 KB. We can decrease the loading time of a Visualforce page by reducing the view state size.
- Try to use the `transient` keyword in custom controller as much as possible. The state is not maintained for transient instance variables. If a particular instance is used only in the page request, then it must not be a part of view state. It will help to reduce the view state size.
- When using an SOQL query to refer data of a particular object, use only the relevant data in the SOQL query.

- When designing the Visualforce page, do not overload the page with excessive functionality and more data. Overloaded pages will increase the view state, page size, heap size, and risk hitting the governor limits for the view state.
- To decrease the loading time of a Visualforce page:
 - Do not use SOQL queries in getter methods
 - Frequently-used or global data must be cached
 - Reduce the number of records displayed in the page by using the built-in pagination of `standardSetControllers` or limiting the data in SOQL queries
- We can increase the time interval for calling the Apex controller by using the `<apex:actionPoller>` component to reduce the delays in multiple concurrent requests.
- Use the SOQL `OFFSET` to implement the pagination of a specific subset of results within SOQL.
- If we have large quantities of read-only data in an organization, then we must use a custom object or custom setting to store that data.
- Use `<apex:repeat>` to iterate over large collections.
- Do not hardcode pick list values in the Visualforce page, and use the controller to add them to a `selectOption` list.
- Use the lazy loading approach to reduce or delay the loading of data according to the essentiality. In lazy loading, the essential features will be loaded first and others will be delayed until the user's action. To lazy load, we can:
 - Use the `reRender` attribute to perform a partial page load
 - Use JavaScript remoting to call actions in the controller
- When using CSS in Visualforce pages, we have to be careful. The performance of Visualforce is directly affected by the optimization of the Visualforce. Here are some tips to increase the performance of CSS:
 - Use separate CSS files and refer to them in the Visualforce page (instead of writing the CSS code in the page itself). This will reduce the file size.
 - Use a single CSS file instead of using multiple CSS files. This will reduce the number of HTTP requests.
 - Refer to CSS files via static resources because it has a in-built caching mechanism.

- When creating pages that have totally customized CSS (not using Salesforce CSS), do not forget to set the attribute of `showHeaders` and `standardStylesheets` of the `<apex:page>` tag to `false`.
- When using JavaScript in Visualforce pages, we have to optimize them to increase the performance of Visualforce. Here are some tips to increase the performance of JavaScript:
 - Use separate JavaScript files and refer to them in the Visualforce page (instead of writing JavaScript in the page itself). This will reduce the size of individual pages taking advantage of browser caching.
 - Use a single JavaScript file instead of using multiple JavaScript files. This will reduce the number of HTTP requests and remove duplicate functions.
 - Use customized versions of JavaScript libraries which include only the required functions. This will reduce the file size.
 - If possible, use the `<script>` tag to include JavaScript in the Visualforce page and place it right before the `</apex:page>` closing tag. This will avoid loading of JavaScript before any other content in the Visualforce page.
 - Eliminate unwanted comments and whitespaces to reduce the file size and for faster downloads.
- Use the `escapeSingleQuotes` method to avoid SOQL and SOSL injection attacks.
- Images frequently play a major role in a Visualforce page. Therefore we have to optimize the usage of images in Visualforce pages using the following tips:
 - Use the CSS sprites instead of individual images. Using sprites, we can combine images (similar sized) into a single file. This will reduce the number of images used in the page and reduce the number of HTTP requests.
 - If possible, try to reduce the use of images and motivate the use of CSS.
 - Use static resources to refer to images in a Visualforce page.



View state cannot be viewed with tools such as Firebug because the view state data is encrypted.

Static resources

Static resources have an in-built caching feature and use the content distribution network built into Salesforce. The following are the advantages of using static resources to refer to CSS files, images, and JavaScript:

- Use a static resource to display the content of another static resource with the action attribute of the `<apex:page>` tag. By doing this we can redirect from a Visualforce page to a static resource. Suppose we have a PDF as a static resource (named as `helpPdf`) and we use that static resource in the action attribute of the `<apex:page>` tag as follows:

```
<apex:page sidebar="false" showHeader="false"
standardStylesheets="false" action="{!URLFOR($Resource.helpPdf)}">
</apex:page>
```

- The `URLFOR` function plays a major role here. The redirection will not work properly without the `URLFOR` function. This is not limited to PDF; we can use any static resource to redirect.

```
<apex:page sidebar="false" showHeader="false"
standardStylesheets="false" action="{!URLFOR($Resource.
helpStaticResource, 'index.htm')}">
</apex:page>
```

Rendering PDFs

When we use components in a Visualforce page and the page is rendered as a PDF, these components do not always work. We must not use components that depend on JavaScript actions and Salesforce standard stylesheets.

- The following components are safe to use in PDF rendering:
 - `<apex:composition>` (as long as the page contains PDF-safe components)
 - `<apex:facet>`
 - `<apex:dataList>`
 - `<apex:define>`
 - `<apex:include>` (as long as the page contains PDF-safe components)
 - `<apex:insert>`
 - `<apex:image>`
 - `<apex:repeat>`
 - `<apex:outputLabel>`

-
- `<apex:outputLink>`
 - `<apex:outputPanel>`
 - `<apex:outputText>`
 - `<apex:page>`
 - `<apex:panelGrid>`
 - `<apex:panelGroup>`
 - `<apex:param>`
 - `<apex:stylesheet>` (as long as the URL isn't directly referencing Salesforce stylesheets)
 - `<apex:variable>`
- The following components can be used with caution in rendering PDF (others are not safe to be used in PDF rendering):
 - `<apex:attribute>`
 - `<apex:column>`
 - `<apex:component>`
 - `<apex:componentBody>`
 - `<apex:dataTable>`

Using component facets

The `<apex:facet>` component is used to specify content in an area of a Visualforce page and it provides information about the data in the parent component. For example, we can use a facet component in the header or footer of a `<apex:dataTable>`. We can override the default facet of a Visualforce component by using the `<apex:facet>` component. The advantages and disadvantages of the facet component are as follows with an example:

- The `<apex:facet>` component cannot be used directly in Apex; it must be a child component of another Visualforce component. We can use that in a dynamic component.
- Facets only allow a single child within the start and close tags.
- The following is an example of the `<apex:facet>` component that is used with the `<apex:dataTable>` component:

```
<apex:page standardController="Item__c">
  <apex:pageBlock>
    <apex:dataTable value="{!item}" var="i">
```

```
        <apex:facet name="caption"><h1>This is {!item.Item_
Name__c}</h1></apex:facet>
        <apex:column>
        <apex:facet name="header">Name</apex:facet>
        <apex:outputText value="{!i.Item_Name__c}"/>
        </apex:column>
        <apex:column>
        <apex:facet name="header">Unit Price</apex:facet>
        <apex:outputText value="{!i.Unit_Price__c }"/>
        </apex:column>
    </apex:dataTable>
</apex:pageBlock>
</apex:page>
```

- We can use the facet component with `<apex:actionStatus>`. It is used to extend the displaying status indicator. This is explained in the following example:

```
<apex:page>
    <apex:form >
        <apex:commandButton value="Facet with action Status"
status="Status" rerender="rerenderBlock"/>
        <apex:pageBlock id="rerenderBlock">
        </apex:pageBlock>
        <apex:actionStatus id="Status">
            <apex:facet name="start">
                
            </apex:facet>
        </apex:actionStatus>
    </apex:form>
</apex:page>
```

Summary

This chapter was dedicated for explaining the best practices of Visualforce developments. In this chapter we became familiar with the best practices to follow in order to avoid unexpected behaviors, reduce the delay experience for accessing component IDs, page block components, controllers and controller extensions, improving Visualforce performance, static resources, rendering PDFs, and using component facets. We have seen the way to improve user experience and coding standards.

Security Tips for Apex and Visualforce Development

Security is an important part of web-based applications. This important part applies for the Force.com applications as well. We create custom pages with Visualforce markup and Apex, and this allows us to provide fully-customized functionality to the client. When we use these programming languages, we must be careful with the security aspects.

The Force.com platform has some in-built security assistance, such as user management, profile management, role hierarchy, **organization wide defaults (OWD)**, permission sets, public groups, sharing settings, field accessibility, password policies, session settings, network access, login access policies, certificate and key management, single sign-on Settings, Auth. Providers, Identity Provider, View Setup Audit Trail, Expire All Passwords, Delegated Administration, Remote Site Settings and HTML Documents and Attachments Settings. But when we create custom pages with Apex and Visualforce, we must be careful because there are many ways to bypass the in-built security defenses on the Force.com platform. There can be general security vulnerabilities as well as Apex and Visualforce specific vulnerabilities.

Security scanning tools

Before we add an application to AppExchange, we have to get the certification for the security aspects. A developer must be aware of these security concerns. There are some tools available to scan our code for security and quality for example, the Force.com Security Source Scanner.

Force.com Security Source Scanner

The Force.com Security Source Scanner is a cloud-based code analysis tool for the Force.com platform. This is a free tool for Force.com developers and the code is scanned on a first-come-first-serves basis. The file size, queue size, and the complexity of the code directly affects the time for getting the scan results. For scanning a particular Salesforce.com user account, we must have the "Author Apex" permission and the particular code must not be contained within a package. We can submit the code for scanning at <http://security.force.com/security/tools/forcecom/scanner>. This tool scans every possible code flow and checks for security vulnerabilities and quality of the Apex code. The Force.com Security Source Scanner can detect the following security vulnerabilities.

- Cross-site scripting
- SOQL injection
- SOSL injection
- Frame spoofing
- Access control issues

The Force.com Security Source Scanner can detect following code and design issues:

- DML statements inside loops
- SOQL/SOSL inside loops
- Not bulkifying Apex methods
- Asynchronous (@future) methods inside loops
- Hardcoding IDs
- Hardcoding `Trigger.new[0]`
- Hardcoding `Trigger.old[0]`
- Referencing static resources
- Queries with no `where` clause or no `LIMIT` clause
- Multiple triggers on the same object

Cross-site scripting (XSS)

Cross-site scripting attacks web applications where there is malicious client-side scripting or HTML. If the web application includes a malicious script, then the attacker can use the web application as an intermediate layer and make the trusted user a victim of the attack. A cross-site scripting weakness occurs when dynamically-generated web pages display invalidated, unfiltered, and non-encoded user input, allowing an attacker to embed malicious scripts into the generated page. This can be leveraged to execute the scripting code as if it came from the site's server on to the computer of anyone who used the site.

The Force.com platform has several methods to protect from XSS attacks, which are as follows:

- **Unescaped output and formulas in Visualforce pages:** There can be Visualforce pages which depend on the user input, and further functionality will proceed with that user input. There are some encoding functions to stop XSS vulnerabilities, which are as follows:
 - **HTMLENCODE:** This function encodes the text and the merged field values to reserved HTML characters. For example, the greater than sign(>) into >.
 - **JSENCODE:** This function encodes the text and merged field values by inserting escape characters.
 - **JSINHTMLENCODE:** This function does the tasks of both the **HTMLENCODE** and **JSENCODE** functions.
 - **URLENCODE:** This function encodes the text and merged field values by replacing illegal characters in URLs. For example, blank spaces are replaced by %20.



All the standard Visualforce components (starting with <apex>) are anti-XSS. They have a filter to stop the XSS attacks. Optionally, we can disable the escape on Visualforce components by setting the value of the escape attribute to false.

Cross-site request forgery (CSRF)

The Web does not, and cannot, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request. In effect, when a server receives a request it has no ability to determine whether that was initiated by a valid user or an attacker, leading to potential escalation of the privilege or theft of data attacks.

The Force.com platform has implemented an anti-CSRF in standard controllers. Each page has random characters as a hidden field. When we load the next page, the validity will be checked and the command will be executed after the value matches with the expected value.

The following code has bypassed the anti-CSRF controls in a custom method called `AutoRun`. There aren't any in-built anti-CSRF controls for such scenarios in the Force.com platform. There are workarounds that will add an intermediate confirmation page before executing the action and shortening the idle session timeout for an organization:

```
<apex:page controller="SecurityIssuesController" sidebar="false"
action="{!AutoRun}">

public class SecurityIssuesController{
public Pagereference AutoRun() {
    Id id = ApexPages.currentPage().getParameters().get('id');
    Item__c singleItem = [select id, Name FROM Item__c WHERE id = :id];
    delete singleItem;
    return null;
}
}
```

SOQL injection

The most popular injection attacks occur when the user's input is directly involved with the query or command. Therefore, the attacker can pass an untrusted date to execute a particular functionality or command. Then the attacker will get the access to unauthorized data.

Apex uses SOQL as the query language and it has limited functionality than SQL. But the SOQL injection attacks are similar to SQL injection attacks. The Salesforce.com users are willing to put their sensitive data into Salesforce because Salesforce.com is a secure platform. Therefore, when we build custom pages and custom controllers, we must pay more attention to prevent such attacks. In Force.com, SOQL injections occur with dynamic SOQL queries.

Dynamic SOQL is used to create the SOQL query string at the runtime of Apex code and allows us to build more flexible functionality (for example, the search functionality which depends on the user's input). Using the `Database.query(queryString)` method, we can create dynamic queries that return a single `sObject` or a list of `sObjects`. The SOQL injection can be implemented in Apex if the application proceeds with the user's input to build a dynamic SOQL and we haven't handled the input properly.

The Force.com platform provides a method called `escapeSingleQuotes` to prevent SOQL injections. Using this method, we can handle the user's input by adding the escape character (`\`) to all single quotations in the user input string. Basically, this method considers all the single quotation as enclosing strings instead of database commands.

The following example illustrates the SOQL injection's vulnerability in Apex. This query returns order records which are not delivered and the customer's name (`cusName`) for the specific order is found according to the user input.

```
String queryString = 'SELECT Id FROM Order__c WHERE (Delivered__c = false and Customer__r.Name like \'%' + cusName + '%')';
```

If the user input is `chamil`, the executing query string would be as follows:

```
queryString = SELECT Id FROM Order__c WHERE (Delivered__c = false and Customer__r.Name like '% chamil %')
```

That's a clean input. But the problem is that users can enter malicious inputs, for example, `chamil%' or Customer__r.Name like '`. Then the query string would look as follows:

```
queryString = SELECT Id FROM Order__c WHERE (Delivered__c = false and Customer__r.Name like '%chamil%' or (Name like '%'))
```

In this case, the result of the query will not return selective orders but will deliver all the orders from the database. This is the impact of SOQL injections. There is a way to protect from such SOQL injection attacks. We can use a string variable to assign the user input to, and add that particular variable to the dynamic query. The following is the fixed code snippet for the preceding vulnerability:

```
String userInput = '%' + cusName + '%';
String queryString = 'SELECT Id FROM Order__c WHERE (Delivered__c = false and Customer__r.Name like: userInput)';
```


Data access control

The Force.com platform allows us to configure object permissions (read, create, edit, and delete) and create data sharing rules. We can implement security controls using those features. The standard controllers adhere to these security settings. But the custom controllers and controller extensions can access all the data during the execution. This is the default behavior, but we can control the data access from Apex classes using the `with sharing` keyword. The keyword is used as follows:

```
public with sharing class ExampleController {
    public void methodOne()
    {
        List<Item__c> = [Select Id, Name FROM Item__c WHERE Id IN:
itemIds];
    }
}
```

The `with sharing` keyword forces the Apex class to consider the security sharing permissions of the logged user.

Summary

We have learned the importance of securing an application. We became familiar with the possible vulnerabilities, solution for those vulnerabilities, and security scanning tools.

Every start has an end, and thus we have reached the end of the book. We have covered the most important topics that will help you to improve the knowledge of Visualforce development. Further, you can use Force.com resources such as the Force.com discussion board (you can obtain help on technical issues), by using `#askforce` on Twitter and <https://blogs.developerforce.com>.

May the force be with you!

Index

Symbols

`$Component` global variable 88
`<apex:actionFunction>` component 17
`<apex:actionPoller>` component 17, 90
`<apex:actionStatus>` component 94
`<apex:actionSupport>` component 17
`<apex:chart>` component 67
`<apex:commandButton>` component 17
`<apex:commandLink>` component 17
`<apex:dataTable>` component 93
`<apex:detail>` component 38, 51
`<apex:facet>` component 93
`<apex:form>` tag 89
`<apex:inputField>` component 38
`<apex:outputPanel>` component 88
`<apex:pageBlock>` component 38, 51
`<apex:pageBlockSectionItem>` component 88
`<apex:pageBlockTable>` component 38, 51
`<apex:page>` component 17
`<apex:page>` tag 16, 92
`<apex:pieSeries>` component 67
`<apex:relatedList>` component 51
`<apex:repeat>` component 90
`<apex:stylesheet>` tag 40
`@RemoteAction` method 71

A

action methods 31
action methods, standard controller
cancel 18
delete 17
edit 17
list 18

quicksave 17
save 17
action methods, standard list controller
about 21
cancel 21
first 21
last 21
List 21
next 21
previous 21
quicksave 21
save 21
addField() method 59
addFields() method 59
Apex 52
Apex classes 8
Apex language 9
Apex Maps
about 61
referencing 61, 62
architecture, Visualforce 9-11
AutoRun method 98

B

BlackBerry configurations, Salesforce
Mobile 76

C

cancel method 18, 21
chart data, providing
about 68
controller method, using 69
JavaScript array, using 70
JavaScript function, using 69
Cloud 7

- cloud computing 7
- complex chart example, Visualforce charting 71-73
- component facets
 - using 93, 94
- component IDs
 - accessing 88
- components, Visualforce
 - <apex:actionFunction> 17
 - <apex:actionPoller> 17
 - <apex:actionSupport> 17
 - <apex:commandButton> 17
 - <apex:commandLink> 17
 - <apex:page> 17
- controller 8, 15
- controller extension
 - about 26
 - building 27, 28
- controller extension development
 - best practices 89
 - considerations 36
- controller methods
 - about 28
 - action methods 31
 - getter methods 29
 - setter methods 30
 - used, for providing chart data 69
- controllers development
 - best practices 89
- cross-platform compatible pages
 - developing 81, 82
- Cross-site request forgery. *See* CSRF
- Cross-site scripting attacks. *See* XSS attacks
- CSRF 98
- CSS 9, 37
- custom attributes, Visualforce 54-56
- custom component, Visualforce
 - about 51, 52
 - creating 52-54
 - using 52-54
- custom controllers, Visualforce 54-56
 - about 23
 - building 23-26
 - considerations 36
- custom objects
 - dynamic references, using with 57-61
- custom styles 38-41

D

- data access control 100
- Data Manipulation Language (DML) 36
- delete method 17
- development tools, Visualforce
 - about 13
 - Eclipse plugin, for Force.com 13
 - Force.com IDE 13
 - Visualforce editor pane 13
- dynamic bindings
 - about 58
- dynamic references
 - using, with custom objects 57-61
 - using, with standard objects 57-61
- dynamic Visualforce binding 57

E

- Eclipse plugin for Force.com 13
- edit method 17
- encoding functions, for stopping XSS vulnerabilities
 - HTMLENCODE 97
 - JSENCODE 97
 - JSINHTMLENCODE 97
 - URLENCODE 97
- escapeSingleQuotes method 91, 99

F

- field sets
 - about 64
 - working with 64
- first method 21
- Force.com
 - about 8
 - MVC model 8
- Force.com IDE 13

G

- getChartData() method 67
- getComplexChartData() method 71
- getter methods 29

H

HTML 4.01 49

HTML5

about 9, 49

and Visualforce pages 49

HTMLENCODE function 97

I

iPhone configurations, Salesforce Mobile 76

J

JavaScript

about 9, 37, 42

using, in Visualforce pages 42

Visualforce components, accessing 42, 43

JavaScript array

used, for providing chart data 70

JavaScript function

used, for providing chart data 69

Javascript remoting

about 44

for Apex controller 44, 45

jQuery

about 9, 47

using, in Visualforce pages 47, 48

JSENCODE function 97

JSINHTMLENCODE function 97

L

last method 21

list method 18

List method 21

Lists

about 61

referencing 61, 62

M

Manage Mobile Configurations permission
86

Mobile configuration 76

mobileforce.device.close() function 83

mobileforce.device.getLocation() function
83

mobileforce.device.syncClose() function 83

mobileforce.device.sync() function 83

mobile-ready Visualforce tab

building 85

model 8

MVC model (Model View Controller)

about 8

controller 8

model 8

view 8

N

next method 21

O

organization wide defaults (OWD) 95

P

page block components 88

pagination feature 21, 22

PDFs rendering 92, 93

performance

improving, of Visualforce 89-91

platforms as services (PaaS) 7

previous method 21

Q

quicksave method 17, 21

R

relationships 58

S

Salesforce.com 7

Salesforce Mobile

about 75

BlackBerry configurations 76

capabilities 77

iPhone configurations 76

limitations 77

Salesforce styles 38

save method 17, 21

- Save method** 16
- Scalable Vector Graphics (SVG)** 66
- security** 95
- security scanning tools**
 - about 95
 - Security Source Scanner 96
- Security Source Scanner** 96
- setter methods** 30
- sObject** 16
- SOQL injection** 98, 99
- SOQL query** 59
- StandardController object** 58
- standard controllers**
 - about 16
 - example 19
 - using, with Visualforce page 16
- standard list controller**
 - about 20
 - using, with Visualforce 20
- standard objects**
 - dynamic references, using with 57-61
- StandardSetController object** 58
- static resources** 92
- styleClass attribute** 38
- styles, Visualforce**
 - custom styles 38-41
 - Salesforce styles 38

T

- tag-based markup language** 9
- transient keyword**
 - about 35
 - using 35

U

- URLENCODE function** 97
- URLFOR function** 92

V

- validation rules** 34
- view** 8
- Visualforce**
 - about 7, 8, 52
 - advantages, for developer 11, 12
 - architecture 9-11

- custom attributes 54-56
- custom components 51, 52
- custom controllers 23, 54-56
- development tools 13
- performance, improving 89-91

Visualforce charting

- about 65
- complex chart example 71, 73
- considerations 66
- limitations 66
- working 66-68

Visualforce components

- accessing, in JavaScript 42, 43

Visualforce editor pane

Visualforce Mobile

- using 78

Visualforce pages

- about 87
- and HTML5 49
- developing 78
- JavaScript library, using 83
- JavaScript, using in 42
- jQuery, using in 47, 48
- mobile configuration, creating 85
- mobile-ready Visualforce tab, building 85
- mobilizing 78
- order, of execution 32-34
- standard controller, using with 16
- standard list controller, using with 20
- styling 38
- working with large datasets 31, 32

Visualforce pages, for BlackBerry

- best practices 78, 79
- considerations 80, 81

Visualforce pages, for iPhone

- best practices 78, 79
- considerations 80

W

- with sharing keyword 89, 100

X

- XSS attacks** 97



Thank you for buying **Visualforce Developer's Guide**

About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

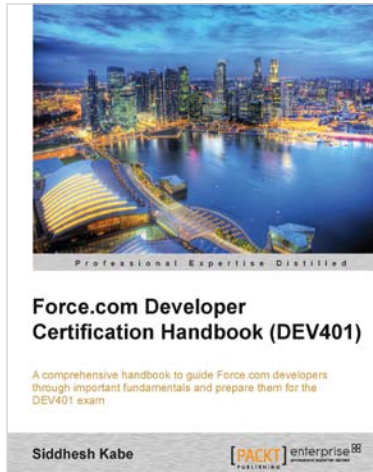
About Packt Enterprise

In 2010, Packt launched two new brands, Packt Enterprise and Packt Open Source, in order to continue its focus on specialization. This book is part of the Packt Enterprise brand, home to books published on enterprise software – software created by major vendors, including (but not limited to) IBM, Microsoft and Oracle, often for use in other corporations. Its titles will offer information relevant to a range of users of this software, including administrators, developers, architects, and end users.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

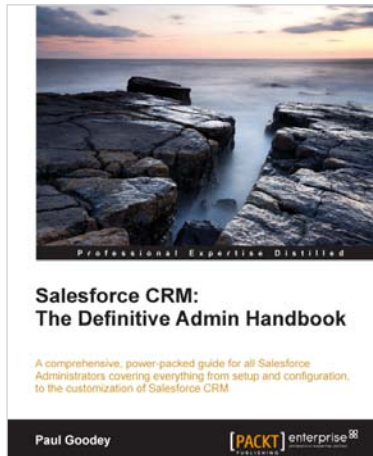


Force.com Developer Certification Handbook (DEV401)

ISBN: 978-1-84968-348-7 Paperback: 280 pages

A comprehensive handbook to guide Force.com developers through important fundamentals and prepare them for the DEV401 exam

1. Simple and to-the-point examples that can be tried out in your developer org
2. A practical book for professionals who want to take the DEV 401 Certification exam
3. Sample questions for every topic in an exam pattern to help you prepare better, and tips to get things started



Salesforce CRM: The Definitive Admin Handbook

ISBN: 978-1-84968-306-7 Paperback: 376 pages

A comprehensive, power-packed guide for all Salesforce Administrators covering everything from setup and configuration, to the customization of Salesforce CRM

1. Get to grips with tips, tricks, best-practice administration principles, and critical design considerations for setting up and customizing Salesforce CRM
2. Master the mechanisms for controlling access to, and the quality of, data and information sharing
3. Take advantage of the only guide with real-world business scenarios for Salesforce CRM

Please check www.PacktPub.com for information on our titles

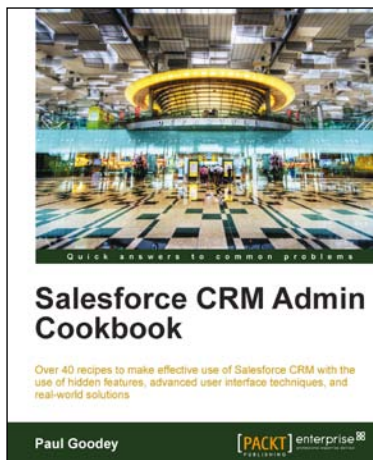


Force.com Tips and Tricks

ISBN: 978-1-84968-474-3 Paperback: 224 pages

A quick reference guide for administrators and developers to get more productive with Force.com

1. Tips and tricks for topics ranging from point-and-click administration, to fine development techniques with Apex & Visualforce
2. Avoids technical jargon, and expresses concepts in a clear and simple manner
3. A pocket guide for experienced Force.com developers



Salesforce CRM Admin Cookbook

ISBN: 978-1-84968-424-8 Paperback: 266 pages

Over 40 recipes to make effective use of Salesforce CRM with the use of hidden features, advanced user interface techniques, and real-world solutions

1. Implement advanced user interface techniques to improve the look and feel of Salesforce CRM
2. Discover hidden features and hacks that extend standard configuration to provide enhanced functionality and customization
3. Build real-world process automation, using the detailed recipes to harness the full power of Salesforce CRM