

# **Force.com Cookbook**

## **Code Samples and Best Practices**

Edited by Carol Franger

With contributions by  
additional members of the salesforce.com  
Technology and Services organizations

## **Force.com Cookbook**

© Copyright 2000-2010 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.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN: 978-0-9789639-2-7

Recipe authors: Stefanie Anderson, Steven Anderson, Mysti Berry, Michelle Chapman-Thurber, Phil Choi, Leah Cutter, Carol Franger, Mark Leonard, Chris McGuire, Teresa Talbot, Garen Torikian, and Joanne Ward. Additional contributors Pierpaolo Bergamo, Dave Carroll, Simon Fell, Steve Fisher, Chris Fry, Richard Greenwald, Larry Jovanovic, Andrea Leszek, Markus Spohn, Nick Tran, Jason Wei, and Craig Weissman. Additional thanks to Andrew Albert, Grant Anderson, Gavin Austin, Eric Bezbar, Manoj Cheenath, Bulent Cinarkaya, Bill Eidson, Matthew Friend, Adam Gross, Michelle Jowitt, Paul Kopacki, Sarah Marovich, Reena Mathew, Taggart Matthiesen, Yudi Nagata, Kavindra Patel, Igor Pesenson, Vahn Phan, Varadarajan Rajaram, Geri Rebstock, Bhavana Rehani, EJ Rice, Jim Rivera, Emad Salman, Mary Scotton, Jerry Sherman, Sagar Wanaselja, Jill Wetzel, and Sarah Whitlock for their advice and support. Special thanks to the editor and lead writer of the First Edition, Caroline Roth, and editor of the Second Edition, Mysti Berry.

# Table of Contents

<b>Welcome.....</b>	<b>1</b>
About This Book.....	1
Intended Audience.....	2
Conventions.....	2
The Sample Recruiting App.....	3
Additional Resources.....	4
Sending Feedback.....	7
Understanding the Force.com Platform.....	7
Using Force.com Tools and Technologies.....	9
Point-and-Click Setup Tools.....	9
Visualforce.....	12
Apex.....	12
Force.com Sites.....	13
Salesforce Mobile.....	13
The Web Services API.....	13
The Bulk API.....	14
The Force.com Migration Tool.....	14
Force.com IDE.....	14
The Metadata API.....	14
<b>Chapter 1: Searching and Querying Data.....</b>	<b>16</b>
Using the Enterprise WSDL and the AJAX Toolkit to Examine Your Data Model.....	18
Using SqqlXplorer to Examine Your Data Model.....	24
Choosing Between SOQL and SOSL.....	26
Querying Multiple Related Objects Using Relationship Queries.....	27
Finding a Contact, Lead, or Person Account.....	29
Retrieving Data Based on a Relative Date.....	31
Finding Search Data Based on Division.....	32
Previewing Query Results.....	33
Sorting Query Results.....	34

Viewing Tags.....	37
Viewing Records with Tags.....	38
Writing Shorter Queries Using Outer Joins.....	39
Making Apex Work in any Organization.....	40
<b>Chapter 2: Managing Workflow and Approvals.....</b>	<b>44</b>
Managing Large Opportunities Using Time-Based Workflow.....	45
Managing Lost Opportunities Using Workflow.....	47
Using Workflow to Notify Case Contact for Priority Cases.....	48
Using Workflow to Add Account Names to Opportunity Names.....	49
Requiring Parallel Approvals for Large Campaigns.....	50
Using a Matrix-Based Dynamic Approval Process.....	54
Sending Outbound Messages with Workflow.....	56
Tracking Outbound Messages from Workflow.....	59
Updating a Field on a Parent Record.....	60
<b>Chapter 3: Customizing the User Interface.....</b>	<b>62</b>
Overriding a Standard Button.....	64
Creating a Button with Apex.....	66
Creating a Consistent Look and Feel with Static Resources.....	67
Formatting a Currency.....	68
Building a Table of Data in a Visualforce Page.....	69
Building a Form in a Visualforce Page.....	71
Creating a Wizard with Visualforce Pages.....	72
Creating Custom Help.....	81
Creating a Custom Visualforce Component.....	82
Overriding a Standard Page.....	85
Redirecting to a Standard Object List Page.....	87
Dynamically Updating a Page.....	88
Overriding a Page for Some, but not All, Users.....	93
Referencing an Existing Page.....	96
Defining Skeleton Visualforce Templates.....	98
Creating Tabbed Accounts.....	102
Adding CSS to Visualforce Pages.....	104
Editing Multiple Records Using a Visualforce List Controller.....	107
Selecting Records with a Visualforce Custom List Controller.....	110

<b>Chapter 4: Displaying Data and Modifying Data Actions.....</b>	<b>112</b>
Creating a Many-to-Many Relationship.....	114
Storing and Displaying Confidential Information.....	118
Averaging Aggregated Data.....	121
Displaying Fields from a Related Record on a Detail Page.....	123
Blocking Record Creation with Cross-Object Validation Rules.....	126
Validating Data Based on Fields in Other Records.....	129
Using Query String Parameters in a Visualforce Page.....	132
Using AJAX in a Visualforce Page.....	134
Using Properties in Apex.....	136
Mass Updating Contacts When an Account Changes.....	140
Bulk Processing Records in a Trigger.....	142
Using Batch Apex to Reassign Account Owners.....	143
Controlling Recursive Triggers.....	149
Comparing Queries Against Trigger.old and Trigger.new.....	152
Preventing Duplicate Records from Saving.....	153
Creating a Child Record When a Parent Record is Created.....	159
Using Custom Settings to Display Data.....	160
Using System.runAs in Test Methods.....	170
Integrating Visualforce and Google Charts.....	172
Using Special Characters in Custom Links.....	177
<b>Chapter 5: Creating Public Websites.....</b>	<b>179</b>
Registering a Custom Domain for Your Force.com Site.....	180
Using Force.com Site-Specific Merge Fields.....	182
Customizing the Look and Feel of Your Force.com Site.....	184
Adding a Feed to Your Force.com Site.....	186
Creating a Sitemap File.....	188
Creating a Web-to-Lead Form for Your Force.com Site.....	190
<b>Chapter 6: Integrating with Other Applications.....</b>	<b>193</b>
Retrieving Information from Incoming Email Messages.....	194
Creating Records from Information in Incoming Email Messages.....	196
Retrieving Email Attachments and Associating Them with Records.....	197
Creating Email Templates and Automatically Sending Emails.....	198
Email Recipes—Complete Code Example.....	202

Updating Salesforce.com Data in the Mobile Application.....	204
Retrieving a User's Location from a GPS-enabled Phone.....	211
Enabling Single Sign-On with the Force.com Platform.....	216
Implementing Single Sign-On for Clients.....	222
<b>Chapter 7: Integrating Applications with the API and Apex.....</b>	<b>227</b>
Setting Up Your Salesforce.com Web Services API Applications.....	228
Select a Development Language.....	228
Create an Integration User.....	228
Select a WSDL.....	229
Generate a WSDL Document.....	230
If You Use the Partner WSDL.....	231
Log In to and Out of the API.....	233
Manage Sessions.....	236
Change the Session Timeout Value.....	237
Implementing the query()/queryMore() Pattern.....	238
Batching Records for API Calls.....	239
Using a Wrapper Class for Common API Functions.....	242
Building a Web Portal with Salesforce.com Data.....	255
Add and Remove Tags on a Single Record.....	261
Add and Remove Tags on Multiple Records.....	263
Updating Tag Definitions.....	265
<b>Force.com Platform Glossary.....</b>	<b>267</b>
<b>Index.....</b>	<b>306</b>

# Welcome

---

Congratulations! You're part of a growing movement of innovative application developers who are curious about the future of computing, and who no longer want to accept the status quo. Maybe your organization just purchased Salesforce.com licenses, or maybe you've been using Salesforce.com for a while and want to extend its capabilities. Maybe you've got a brilliant business idea and are looking for the best and fastest way to start making money, or maybe you're just curious about this thing called Apex and want to keep your skill set up to date with the latest technology.

No matter what angle you're coming from, this book helps application developers leverage the power of the Force.com cloud platform to build fully functional, integrated Web applications that free you and your organization from the drudgery of maintaining your own software and hardware stacks. Instead, you can spend your time and money on the ideas and innovations that make your business applications special, whether you're a lone developer looking for your first round of venture funding or part of a multi-billion-dollar company with hundreds of thousands of employees.

## About This Book

This book provides many recipes for using the Web Services API, developing Apex scripts, and creating Visualforce pages. Written by developers, the *Force.com Cookbook: Code Samples and Best Practices* helps developers become familiar with common Force.com programming techniques and best practices.

To get the most out of the recipes in this book, make sure you understand the experience you should have, and the tools to supplement these recipes:

- [Intended Audience](#) on page 2
- [Conventions](#) on page 2

- [The Sample Recruiting App](#) on page 3
- [Additional Resources](#) on page 4
- [Sending Feedback](#) on page 7



**Note:** This book indicates the recipes for which salesforce.com Training & Certification has provided example code or other information. To enroll in courses that provide even more information and practical experience, see [www.salesforce.com/training](http://www.salesforce.com/training).

## Intended Audience

Developers who are already familiar with the point-and-click capabilities of the Force.com platform can most easily implement the recipes in this book. Before working with the recipes you find here, you should be familiar with the concepts, techniques, and best practices described in *Force.com Fundamentals: An Introduction to Custom Application Development in the Cloud*, available on the Developer Force website at [developer.force.com/books/fundamentals](http://developer.force.com/books/fundamentals).

To get the most out of this book, you should also have experience with at least one of the following:

- HTML and JavaScript
- Java
- C#.NET
- VB.NET
- PHP
- Python
- Ruby
- Perl
- Any other Web-services-enabled programming language

## Conventions

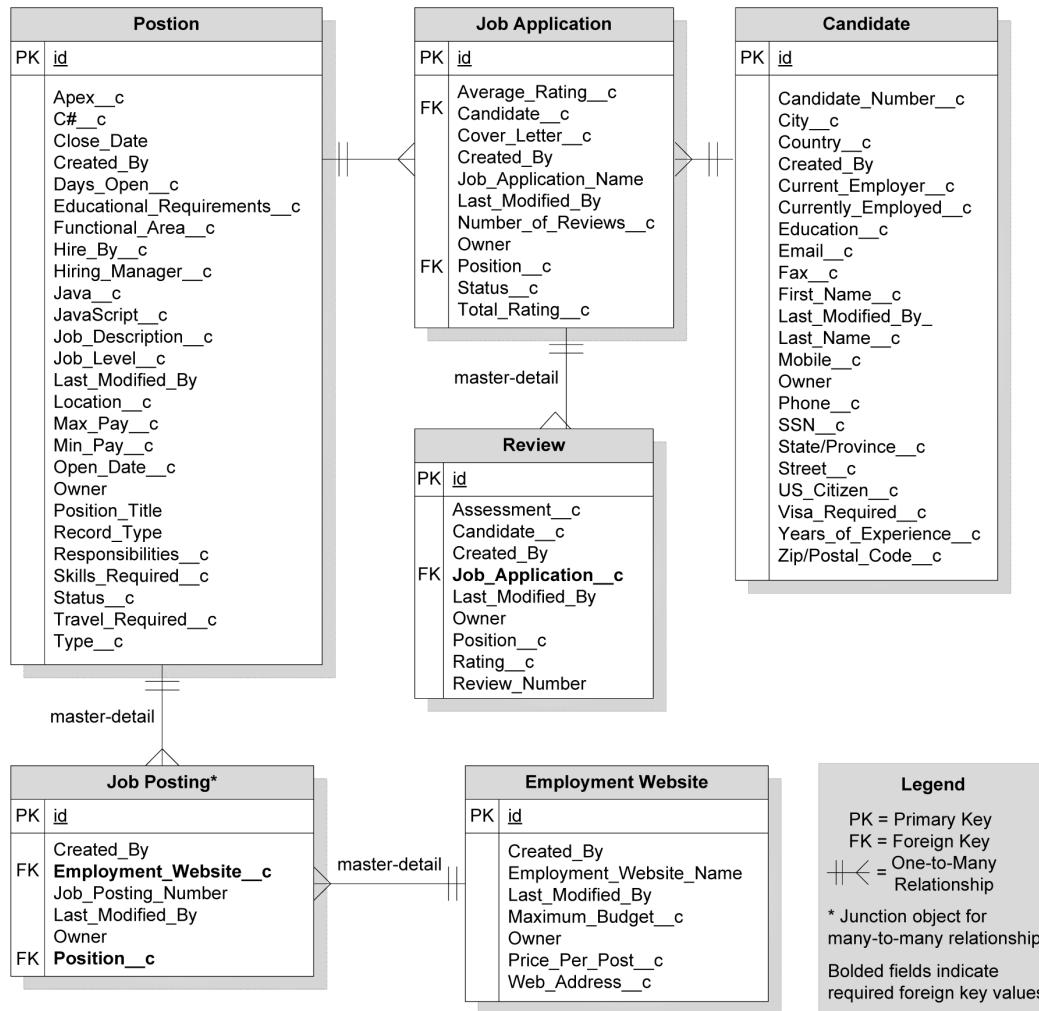
This book uses the following typographical conventions:

Convention	Description
<code>SELECT Name FROM Account</code>	In an example, Courier font indicates items that you should type as shown. In a syntax statement, Courier font also indicates items that you should type as shown, except for question marks and square brackets.

Convention	Description
SELECT <i>fieldname</i> FROM <i>objectname</i>	In an example or syntax statement, italics represent variables. You supply the actual value.
?	In a syntax statement, the question mark indicates the element preceding it is optional. You may omit the element or include one.
WHERE [ <i>conditionexpression</i> ]	In a syntax statement, square brackets surround an element that may be repeated up to the limits for that element. You may omit the element, or include one or more of them.

## The Sample Recruiting App

Some of the recipes in this book require a custom app for illustration. In these cases, this book uses the custom Recruiting app that was developed as part of *Force.com Fundamentals: An Introduction to Custom Application Development in the Cloud*. The schema for this Recruiting app is pictured in the following entity-relationship diagram and can be downloaded from [developer.force.com/books/cookbook](http://developer.force.com/books/cookbook).



**Figure 1: Schema for the Sample Recruiting App**

## Additional Resources

A variety of resources are available to supplement this book.

### Documentation

### Developer Force Technical Library

The Developer Force Technical Library, at [wiki.developerforce.com/index.php/Documentation](http://wiki.developerforce.com/index.php/Documentation), is your resource for information about developing on the Force.com platform. Here you can learn Force.com fundamentals and advanced programming techniques, get started with Visualforce or Apex, learn how to write SOQL and SOSL queries, dig into the Web services and metadata API, and more.

### Force.com Point-and-Click Functionality

- Access the Help & Training window by clicking **Help** or **Help & Training** in the upper-right corner of any Salesforce.com page. Alternatively, access a context-sensitive view of the Help & Training window by clicking **Help for this Page** on the right side of any page title bar, or the help link on the right side of any related list.
- Review white papers, multimedia presentations, and other documentation in the Application Framework section of the Developer Force website at [wiki.developerforce.com/index.php/Application\\_Framework](http://wiki.developerforce.com/index.php/Application_Framework).
- Review tips and best practices at [www.salesforce.com/community](http://www.salesforce.com/community).

### Visualforce

- Read the *Visualforce Developer's Guide*, available at [www.salesforce.com/us/developer/docs/pages/index.htm](http://www.salesforce.com/us/developer/docs/pages/index.htm).
- Review white papers, multimedia presentations, and other documentation in the Visualforce section of the Developer Force website at [wiki.apexdevnet.com/index.php/Visualforce](http://wiki.apexdevnet.com/index.php/Visualforce).

### Apex

- Read the *Force.com Apex Developer's Guide*, available at [www.salesforce.com/us/developer/docs/apexcode/index.htm](http://www.salesforce.com/us/developer/docs/apexcode/index.htm).
- Review white papers, multimedia presentations, and other documentation in the Apex section of the Developer Force website at [wiki.developerforce.com/index.php/Apex](http://wiki.developerforce.com/index.php/Apex).

### Web Services API

- Read the *Force.com Web Services API Developer's Guide*, available at [www.salesforce.com/apidoc](http://www.salesforce.com/apidoc).
- Review whitepapers, multimedia presentations, and other documentation in the API section of the Developer Force website at [wiki.apexdevnet.com/index.php/Web\\_Services\\_API](http://wiki.apexdevnet.com/index.php/Web_Services_API).

### Metadata API

Read the *Force.com Metadata API Developer's Guide*, available at [www.salesforce.com/us/developer/docs/api\\_meta/index.htm](http://www.salesforce.com/us/developer/docs/api_meta/index.htm).

## Bulk API

Read the *Force.com Bulk API Developer's Guide*, available at [www.salesforce.com/us/developer/docs/api\\_asynch/index.htm](http://www.salesforce.com/us/developer/docs/api_asynch/index.htm).

## Training Courses

Training classes are also available from salesforce.com Training & Certification. You can find a complete list of courses at [www.salesforce.com/training](http://www.salesforce.com/training).

## Free Developer Edition Account

If you have not already done so, visit the Developer Force website at [developer.force.com](http://developer.force.com) and click **Join Now** to sign up for a free, two-user Developer Edition organization.

## Integrated Development Tools

The Force.com platform provides developer tools that are tightly integrated with the platform. You may wish to select a tool and become familiar with it before working with the recipes in this book.

- The Force.com Migration Tool is generally available. To access it, log in to your Salesforce.com organization and select **Setup** ▶ **Develop** ▶ **Tools** and click **Force.com Migration Tool**.
- The Force.com IDE is the world's first integrated development environment for cloud computing. Based on Eclipse technology, the Force.com IDE provides professional developers and development teams the tools to code, test, deploy, and version Force.com components, including Apex, Visualforce, custom objects, layouts, and more. For information about this tool, see [wiki.apexdevnet.com/index.php/Force.com\\_IDE](http://wiki.apexdevnet.com/index.php/Force.com_IDE).
- The latest version of AJAX Tools, an AppExchange package alternative to the Force.com IDE, includes syntax-highlighting, a tool for exploring the data model of your organization, and code samples. To download the latest version, go to [sites.force.com/appexchange/listingDetail?listingId=a0330000002foeKAAQ](http://sites.force.com/appexchange/listingDetail?listingId=a0330000002foeKAAQ).
- For Mac users, SoqlXplorer provides metadata exploration, a SOQL query tester, and a graphical schema view for examining object relationships (a piece of functionality that's only available on the Mac OS X platform!). Download SoqlXplorer from Simon Fell's PocketSOAP website at [www.pocketsoap.com/osx/soqlx](http://www.pocketsoap.com/osx/soqlx). After the download automatically extracts itself, drag the SoqlXplorer icon to your Applications folder to complete the installation.

For other great Salesforce.com tools and utilities built exclusively for Mac OS X, see [www.pocketsoap.com/osx](http://www.pocketsoap.com/osx).



**Tip:** Visit IdeaExchange at [ideas.salesforce.com](http://ideas.salesforce.com) and see what users are asking for. IdeaExchange is a forum where salesforce.com customers can suggest new product concepts, promote favorite enhancements, interact with product managers and other customers, and preview what salesforce.com is planning to deliver in future releases. You can use IdeaExchange both to find ideas for new applications, and to post your pet peeves about how the platform works for you.

## Sending Feedback

The authors made every effort to ensure the accuracy of the information contained within this book, but neither they nor salesforce.com assumes any responsibility or liability for any errors or inaccuracies that may appear. If you do find any errors, please send feedback to [docfeedback@salesforce.com](mailto:docfeedback@salesforce.com), or visit the Developer Force discussion boards at [www.salesforce.com/developer/community/index.jsp](http://www.salesforce.com/developer/community/index.jsp).

## Understanding the Force.com Platform

The Force.com platform makes it easy to build applications for cloud computing. It's the world's first platform for building, sharing, and running business applications in the cloud. The Force.com platform is unique among development platforms for several reasons.

### Delivery

The Force.com platform runs in a hosted, multitenant environment. That means you can access any app you build on the platform from anywhere in the world with just an Internet connection and a Web browser. No servers or databases need to be maintained, and no software needs to be installed or upgraded. Instead, salesforce.com provides a hosted environment in which the latest features and functionality are seamlessly available to all users with every new release. And you'll have the peace of mind of knowing that any app you were using or building before the new release will work just as well after the release too, regardless of whether it was a standard CRM app from salesforce.com or a custom app you developed on your own.

As a developer, the platform's multitenant architecture also means that you never have to worry about scaling your apps from one to one thousand or even to one million users—all of the infrastructure to handle such growth is provided free of charge, automatically behind the scenes. That leaves you more time to focus on your business problems and solutions,

rather than anticipating the pressures that increased usage might exert on your apps.

## Distribution

Any app written on the platform has access to a built-in community of potential customers on the AppExchange at <http://sites.force.com/appexchange>. Unlike traditional software, where you have to create an install wizard and find a way to distribute the product, you can easily share and distribute your app on the AppExchange with only a few clicks of the mouse. You can share your apps privately with just the people you want, or you can publish your apps for anyone to download.

If you do publish an app publicly, the community of users on the AppExchange can take your app for a test drive and review comments from other users about how well it worked. Additionally, information about the users who download your app is sent directly to you in the form of a new lead in any Salesforce.com organization that you specify.

When you're ready to release new versions of your app, the AppExchange also helps you communicate and manage the upgrade process for all of your users. You can track which of your customers are on which version of your app, and you never have to worry that your users have broken or deleted any component your app relies on.

## Development

The Force.com platform comes with a wide variety of built-in, point-and-click functionality that help you build your apps faster. Need a way to store data in your app? Define new database objects, fields, and relationships declaratively with the mouse, rather than by composing SQL CREATE statements. Need to control which users have access to different kinds of data? Again, no coding necessary—just use the security and sharing framework to define permissions at different levels of granularity, from individual fields to entire objects and applications. The Force.com platform includes point-and-click tools for everything from string localization to workflow rules and approval processes, from custom reports and dashboards to page layouts and data import wizards—which means you can spend less time recreating the “plumbing” that makes your applications run and more time on the unique functionality that sets your apps apart from your competitor's.

And what happens when you want to go beyond the capabilities of the point-and-click tools the platform provides? The Web Services API, Apex, and Visualforce give you the flexibility you need to build the applications you want. Integrate third-party Web services with embedded mashups, change the logic behind every function with Apex classes and triggers,

and redesign the user interface the way you want with Visualforce. You're limited only by your imagination!

The Force.com platform includes a number of tools that can help you develop apps. These tools allow you to define the data, business logic, and user interface for an application. The recipes in this book focus on these tools.

## Using Force.com Tools and Technologies

The Force.com platform includes a number of tools that can help you build apps. These tools allow you to define the data, business logic, and user interface for an application.

### Point-and-Click Setup Tools

The Force.com platform includes declarative, point-and-click setup tools that allow administrators and developers to quickly build common application components without writing any code. These setup tools allow you to effortlessly build:

#### Data Components

Data components are equivalent to the “model” in the MVC application development paradigm. They include:

- **Custom objects**

Similar to a database table, a Salesforce.com object is a structure for storing data about a certain type of entity, such as a person, account, or job application. Salesforce.com includes over a dozen standard objects that support default apps like Sales and Service & Support, but it also allows you to build custom objects for your own application needs. In Salesforce.com, each object automatically includes built-in features like a user interface, a security and sharing model, workflow processes, search, and much more.

- **Custom fields**

Similar to a column in a database table, a Salesforce.com field is a property of an object, such as the first name of a contact or the status of an opportunity. Salesforce.com fields support over a dozen different field types, such as auto-number, checkbox, date/time, and multi-select picklists.

- **Custom relationships**

Similar to the way primary and foreign keys work in a relational database, a Salesforce.com relationship defines a connection between two objects in which matching values in a specified field in both objects are used to link related data.

- **Field history**

Salesforce.com field history allows you to track changes to fields on a particular object just by selecting a checkbox on a custom object and field definition. Users can then review audit logs for changes to sensitive records without any additional development work.

## **Business Logic Components**

Business logic components are equivalent to the “controller” in the Model-View-Controller (MVC) application development paradigm. They include:

- **Security and permission settings**

Salesforce.com security and permissions tools, such as user profiles, organization-wide defaults, the role hierarchy, sharing rules, and manual sharing, allow you to control the data that users can view and edit, with either broad generalizations or a fine level of detail.

- **Formula fields and validation rules**

Formula fields, default field values, and validation rules allow you to use Excel-like syntax to calculate certain data automatically, maintain data quality, and add custom error messages to your apps.

- **Workflow rules**

Workflow rules are processes triggered by user activity or according to a schedule. These processes can automatically assign tasks to users, send email alerts to multiple recipients, update field values in records, and even generate SOAP messages to external Web services.

- **Approval processes**

Approvals allow you to set up a chain of users who can approve the creation of sensitive types of records, such as new contracts or vacation requests.

- **Email**

Email functionality in Salesforce.com allows you to email contacts, leads, person accounts, and users in your organization directly from account, contact, lead, opportunity, case, campaign, or custom object pages.

## User Interface Components

User interface components are equivalent to the “view” in the MVC application development paradigm. They include:

- **Tabs**

Tabs give users a starting point for viewing, editing, and entering information for a particular object. When a user clicks a tab at the top of the page, the corresponding tab home page for that object appears.

- **Page layouts**

Regardless of whether a particular object has a tab, all objects can be viewed or edited. Page layouts allow you to organize the fields, custom links, related lists, and other components that appear on those pages.

- **Custom views**

Custom views allow users to filter the records they see for a particular object, based on criteria they specify.

- **Reports and Dashboards**

Salesforce.com includes a full-featured report building tool, including custom report types that allow you to view data for any combination of objects, and dynamic dashboards that give users a bird's eye view of their application data.

- **Console**

Salesforce.com Console allows you to set up a page that displays multiple objects at a time, streamlining the user experience. It includes a list view of several different objects at the top of the page, a detail view in the main window, customizable sidebar components, and mini-detail views of related information in a dynamic AJAX-based interface.

With this functionality, app developers can build extensive, full-featured applications that handle many business needs. For a more thorough introduction to the functionality provided

by the platform, read *Force.com Fundamentals: An Introduction to Custom Application Development in the Cloud*, available on the Developer Force website at [developer.force.com/books/fundamentals](http://developer.force.com/books/fundamentals).

## Visualforce

Visualforce is a tag-based markup language that allows developers to develop their own custom interfaces using standard Salesforce.com components. Visualforce pages deliver the ability to create custom pages for your Force.com applications. Visualforce pages includes a set of tags to describe a variety of rich components into your page design. These components bring the full power of the metadata-driven Force.com platform to your pages, while giving you complete freedom to design pages to suit your specific user interface requirements. The components can either be controlled by the same logic that's used in standard Salesforce.com pages, or developers can associate their own logic with a 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 user interface, while developers can work on the business logic that drives the app.

## Apex

Apex is a strongly-typed, object-oriented programming language for executing flow and transaction control statements on the Force.com server in conjunction with database queries, inserts, updates, and deletes. Using syntax that looks like Java and acts like database stored procedures, Apex allows you to add business logic to your applications in a more efficient, integrated way than is possible with the Web Services API.

You can manage and invoke Apex scripts using the following constructs:

- **Classes**

A *class* is a template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code.

Once successfully saved, class methods or variables can be invoked by other Apex scripts, or through the Force.com Web Services API (or AJAX Toolkit) for methods that have been designated with the `webService` keyword.

In most cases, Apex classes are modeled on their counterparts in Java and can be quickly understood by those who are familiar with them.

- **Triggers**

A *trigger* is an Apex script that executes before or after specific data manipulation language (DML) events occur, such as before object records are inserted into the database, or after records have been deleted. Other than Apex Web service methods, triggers provide the primary means for instantiating Apex.

- **Anonymous blocks**

An *anonymous block* is an Apex script that does not get stored in the metadata, but that can be compiled and executed through the use of the `executeanonymous()` API call or the equivalent in the AJAX toolkit.

## Force.com Sites

Force.com sites enables you to create public websites and applications that are directly integrated with your Salesforce.com organization—without requiring users to log in with a username and password. Sites are hosted on Salesforce.com servers, and are built on Visualforce pages.

## Salesforce Mobile

Salesforce Mobile is a Salesforce.com feature that enables users to access their Salesforce.com data from mobile devices running the mobile client application. The Salesforce Mobile client application exchanges data with Salesforce.com over wireless carrier networks, and stores a local copy of the user's data in its own database on the mobile device. Users can edit local copies of their Salesforce.com records when a wireless connection is unavailable, and transmit those changes when a wireless connection becomes available.

Salesforce Mobile works with Apex and Visualforce to extend functionality.

## The Web Services API

Salesforce.com provides programmatic access to your organization's information using a simple, powerful, and secure application programming interface, the Force.com Web Services API (the API). You can use the SOAP-based API to create, retrieve, update or delete records, such as accounts, leads, and custom objects. With more than 20 different calls, the API also allows you to maintain passwords, perform searches, and much more. Use the API in any language that supports Web services.

## The Bulk API

The REST-based Bulk API is optimized for loading or deleting large sets of data. It allows you to insert, update, upsert, or delete a large number of records asynchronously by submitting a number of batches which are processed in the background by Salesforce.com.

The SOAP-based API, in contrast, is optimized for real-time client applications that update small numbers of records at a time. Although the SOAP-based API can also be used for processing large numbers of records, when the data sets contain hundreds of thousands of records it becomes less practical. The Bulk API is designed to make it simple to process data from a few thousand to millions of records.

## The Force.com Migration Tool

The Force.com Migration Tool is a Java/Ant-based command-line utility for moving metadata between a local directory and a Salesforce.com organization. You can use the Force.com Migration Tool to retrieve components, create scripted deployment, and repeat deployment patterns.

## Force.com IDE

The Force.com IDE is a plug-in for the Eclipse IDE that provides special help for developing and deploying Apex classes, Apex triggers, Visualforce pages, and metadata components.

## The Metadata API

Use the Metadata API to retrieve, deploy, create, update or delete customization information, such as custom object definitions and page layouts, for your organization. The most common usage is to migrate changes from a sandbox or testing organization to your production organization. The Metadata API is intended for managing customizations and for building tools that can manage the metadata model, not the data itself. To create, retrieve, update or delete records, such as accounts or leads, use the API to manage your data.

You can modify metadata in *test* organizations on Developer Edition or Sandbox, and then deploy tested changes to *production* organizations on Enterprise Edition or Unlimited Editions. You can also create scripts to populate a new organization with your custom objects, custom fields, and other components.

The easiest way to access the functionality in the Metadata API is to use the Force.com IDE or Force.com Migration Tool. These tools are built on top of the Metadata API and use the standard Eclipse and Ant tools respectively to simplify the task of working with the Metadata API. Built on the Eclipse platform, the Force.com IDE provides a comfortable environment for programmers familiar with integrated development environments, allowing you to code, compile, test, and deploy all from within the IDE itself. The Force.com Migration Tool is ideal if you want to use a script or a command-line utility for moving metadata between a local directory and a Salesforce.com organization.

# Chapter 1

## Searching and Querying Data

### In this chapter ...

- Using the Enterprise WSDL and the AJAX Toolkit to Examine Your Data Model
- Using SoqlXplorer to Examine Your Data Model
- Choosing Between SOQL and SOSL
- Querying Multiple Related Objects Using Relationship Queries
- Finding a Contact, Lead, or Person Account
- Retrieving Data Based on a Relative Date
- Finding Search Data Based on Division
- Previewing Query Results
- Sorting Query Results
- Viewing Tags
- Viewing Records with Tags
- Writing Shorter Queries Using Outer Joins

Truly useful, cloud computing business apps include business logic and processes that help companies run their businesses efficiently. As we've mentioned, the Force.com platform gives you the power to write code and develop components to incorporate business logic, such as data validation, into your app. Pretty much any business process you write for your app will require your code to search, query, and examine sets of records upon which the business process will operate. So what's the best way to do that?

In this chapter, you'll learn how to examine your app's objects, relationships, and fields in a graphical way. You'll also learn the difference between SOQL and SOSL and how to use them to construct queries that examine sets of records in your app. Then you'll see how to use SOQL to query related objects using their relationship associations and how to filter your queries by a relative date or the division of a record. Finally, you'll use the AJAX toolkit to query your data in a command-line environment. These examples and best practices are a great way to get started developing your own queries to manipulate the data in ways that are unique to your app.

You can use the recipes in this chapter without setting up the API, because the tools you'll use are already set up for you. But all the query techniques you learn here can be used if you do work with the API. For more information, see [Integrating Applications with the API and Apex](#) on page 227.

- Making Apex Work in any Organization

# Using the Enterprise WSDL and the AJAX Toolkit to Examine Your Data Model

## Problem

You want to browse through the fields, attributes, and relationships of every object in your Salesforce.com organization, and you're not on a Mac OS X platform.

## Solution

Use the enterprise WSDL and the AJAX Toolkit. No installation is necessary, these tools are available to any organization with API access.



**Tip:** The permissions associated with your login affect the visibility of objects and fields in the enterprise WSDL and the AJAX Toolkit. Be sure that your login has access to the data you need to explore—a user created specifically for API access, with the “Modify All Data” permission but not other administration permissions, typically works best.

First, you can inspect objects and their fields, calls, SOAP headers and status codes (error codes) just by reviewing the WSDL:

1. Navigate to <https://www.salesforce.com>, click the Customer Login tab and log in to your Salesforce.com organization.
2. Click **Setup > Develop > API**.
3. Generate the enterprise WSDL.
  - If you have managed packages installed in your organization, click **Generate Enterprise WSDL**. Salesforce.com prompts you to select the version of each installed package to include in the generated WSDL.
  - Otherwise, right-click **Generate Enterprise WSDL** to save the WSDL to a local directory. In the right-click menu, Internet Explorer users can choose **Save Target As**, while Mozilla Firefox users can choose **Save Link As**.
4. Open the WSDL file you just saved in your favorite XML editor.
5. Scroll through the file. There is an entry for each standard and custom object visible to the logged in user, as well as entries for calls, SOAP headers, and status codes. Notice that each standard object is a complex type that extends `sObject`.

You can find some relationships by inspecting the WSDL. For example, there is a field in the Contact object that contains account IDs:

```
<element name="AccountId" nullable="true" minOccurs="0"
type="tns:ID"/>
```

Thus, it's clear that Contact has a relationship with Account, though you must use the AJAX Toolkit to be sure of the type of relationship.

Next, you can inspect objects and their fields using the AJAX Toolkit:

1. Navigate to the AJAX Toolkit debug shell, using the correct URL for the API version. For example, the following URL is for an organization querying version 15.0:

```
https://na1.salesforce.com/soap/ajax/15.0/debugshell.html
```

You may need to log directly into your organization and cut and paste the filepath.

2. If requested, log in. Otherwise, issue this command in the debug shell to log in:

```
sforce.connection.login("MyName@myISP.com", "myPassword")
```

You'll see a table of information, including the user information. Notice that every request (call) made in the AJAX Toolkit is preceded by `sforce.connection..`. That's the JavaScript library that contains the toolkit.

3. Issue a `describeSObject` call to inspect the object you are curious about:

```
sforce.connection.describeSObject("Contact")
```

()	
activateable	false
createable	true
custom	false
deletable	false
deprecated	false
deprecatedAndHidden	false
keyPrefix	003
label	Contact
labelPlural	Contacts
layoutable	true
mergeable	false
name	Contact
queryable	true
replicable	true
retrieveable	true
searchable	true
triggerable	true
undeletable	false
updateable	true
urlDetail	<a href="https://na1.salesforce.com/{ID}">https://na1.salesforce.com/{ID}</a>
urlEdit	<a href="https://na1.salesforce.com/{ID}/e">https://na1.salesforce.com/{ID}/e</a>
urlNew	<a href="https://na1.salesforce.com/003/e">https://na1.salesforce.com/003/e</a>
childRelationships	(0)
fields	(0)
recordTypeInfos	(0)

**Figure 2: Output from** describeSObject

You can find important information about this object:

**Key Prefix**

The first three characters of the Salesforce.com ID for records of this object type. For example, the ID of any account record always starts with 001. In this example you can see that contact records always start with 003.

**Label and LabelPlural**

The labels that are used to display this object in both singular and plural form.

**URL Detail**

The URLs that can be used to reach detail, edit, and list pages for the object in a Web browser.

**Attributes**

A list of actions that you can perform on the object. See [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_describesobjects\\_describesobjectresult.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_describesobjects_describesobjectresult.htm).

4. To see which objects are children of this object, click the arrows icon next to childRelationships:

childRelationships			
cascadeDelete	childSObject	field	relationship
true	AccountContactRole	ContactId	AccountContact
true	ActivityHistory	WhoId	ActivityHistories
false	Asset	ContactId	Assets
false	Attachment	ParentId	Attachments
true	CampaignMember	ContactId	CampaignMembers
false	Case	ContactId	Cases
true	CaseContactRole	ContactId	CaseContactRole
true	CaseTeamMember	MemberId	null
true	CaseTeamTemplateMember	MemberId	null
false	Contact	ReportsToId	null
true	ContactHistory	ContactId	Histories
true	ContactShare	ContactId	Shares
true	ContactTag	ItemId	Tags
false	Contract	CustomerSignedId	ContractsSigned
true	ContractContactRole	ContactId	ContractContact
true	EmailStatus	WhoId	EmailStatuses
true	Event	WhoId	Events
true	EventAttendee	AttendeeId	null
false	Lead	ConvertedContactId	null
true	Note	ParentId	Notes
true	NoteAndAttachment	ParentId	NotesAndAttachments
true	OpenActivity	WhoId	OpenActivities
true	OpportunityContactRole	ContactId	OpportunityContacts
true	ProcessInstance	TargetObjectId	ProcessInstances
false	ProcessInstanceHistory	TargetObjectId	ProcessSteps
false	SelfServiceUser	ContactId	null
true	Task	WhoId	Tasks

**Figure 3: Child Objects**

In this example, notice that some fields are named WhoId or ParentId instead of the expected ContactId. These names are used when the parent could be the object you are inspecting, in this case Contact, or another object.

The relationships that have been defined on other objects that reference this object as the “one” side of a one-to-many relationship. For example, if you expand the Child Relationships node under the Account object, contacts, opportunities, and tasks are included in this list.



**Note:** Relationships that are defined on this object so that it represents the “many” side of a one-to-many relationship (for example, the Parent Account relationship on the Account object) are included in the list of fields.

- To see the fields that this object contains, click the arrows icon next to fields:

createable	custom	defaultedOnCreate	digits	filterable	idLookup	label	name
false	false	true	0	true	true	Contact ID	Id
false	false	true	0	true	false	Deleted	IsDeleted
false	false	false	0	true	false	Master Record ID	MasterRecordId
true	false	false	0	true	false	Account ID	AccountId
true	false	false	0	true	false	Last Name	LastName
true	false	false	0	true	false	First Name	FirstName
true	false	false	0	true	false	Salutation	Salutation
false	false	false	0	true	false	Full Name	Name
true	false	false	0	true	false	Other Street	OtherStreet
true	false	false	0	true	false	Other City	OtherCity
true	false	false	0	true	false	Other State/Province	OtherState
						Other	Other

**Figure 4: Output from describeSObject fields**

You can find important information about the fields on this object, including the API name for the field and the corresponding Salesforce.com user interface name. You can also see which attributes are set to true for this field. For example, if `creatable` is true, than this field can be created on a record using the API.



**Note:** Standard objects are listed by their standard names, even if you've renamed them.

## Discussion

To have the AJAX Toolkit use a different version of the API, change the version number in the URL. For example, the following URL accesses an organization on instance na1 via version 15.0 of the API:

<https://na1.salesforce.com/soap/ajax/15.0/debugshell.html>

If you change 15 to 14, you'll access the same organization on the same instance with version 14.0 of the API. This may be useful for testing.

### See Also

[Using SoqlXplorer to Examine Your Data Model](#) on page 24

## Using SoqlXplorer to Examine Your Data Model

### Problem

You want to browse through the fields, attributes, and relationships of every object in your Salesforce.com organization, and you're on the Mac OS X platform.

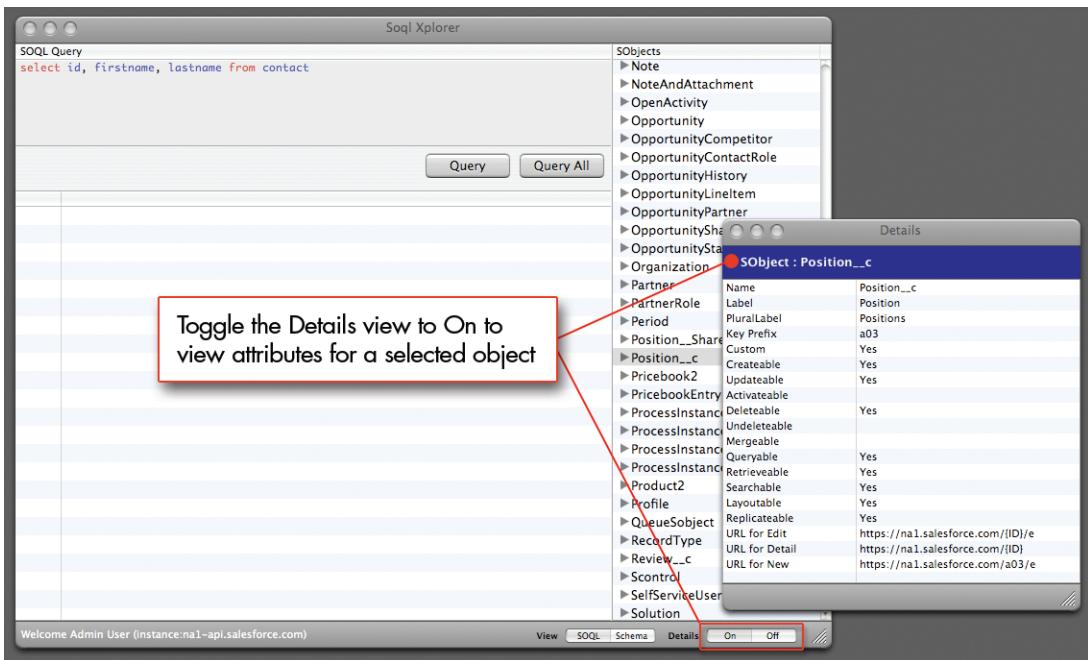
### Solution

For Mac users, SoqlXplorer provides metadata exploration, a SOQL query tester, and a graphical schema view for examining object relationships (a piece of functionality that's only available on the Mac OS X platform!). Download SoqlXplorer from Simon Fell's PocketSOAP website at [www.pocketsoap.com/osx/soqlx](http://www.pocketsoap.com/osx/soqlx). After the download automatically extracts itself, drag the SoqlXplorer icon to your Applications folder to complete the installation.

After installing SoqlXplorer, open the application and log in by entering your standard Salesforce.com username and password and specifying the server to which you want to connect. Choose www.salesforce.com to connect to the normal production servers, or test.salesforce.com to connect to a sandbox organization.

After you click **Login**, SoqlXplorer issues a `describeGlobal()` call to the API to populate the interactive list of objects in the right sidebar.

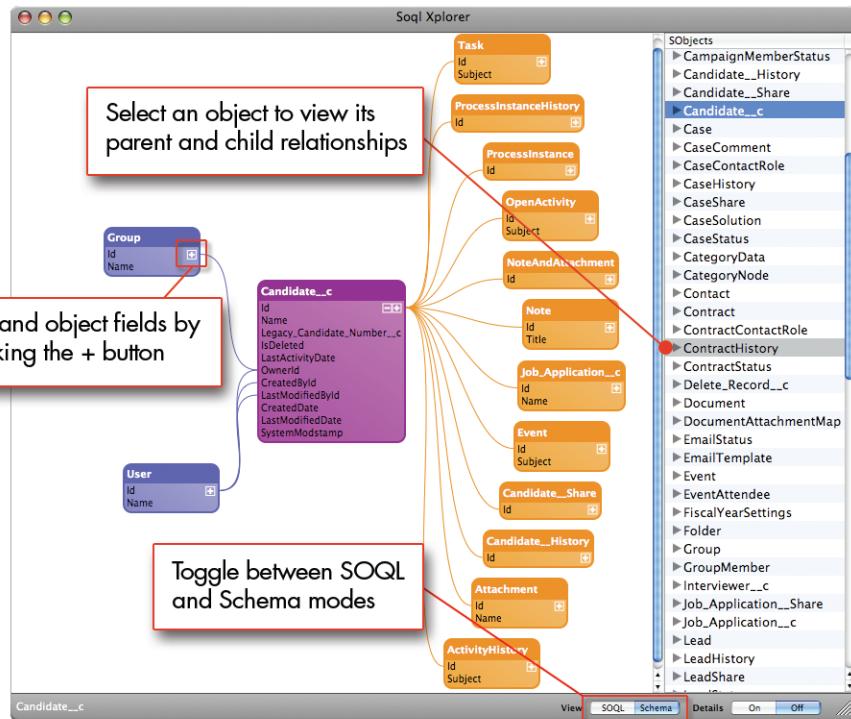
You can expand any object to explore its fields and relationships. To view attributes for an object, toggle the **Details** button to On in the bottom right corner of the window. If you select an object field, the **Details** popup shows properties for the field instead.



**Figure 5: Viewing Object Attributes in SoqlXplorer**

Two views are available in the main window: **SOQL** and **Schema**.

- Use **SOQL** view to open a SOQL query editor where you can construct and execute SOQL queries. The queries you write use syntax-highlighting to improve legibility, and you can double-click an object's name to automatically build a query that selects all available fields. You can also double-click any result data to copy and paste it elsewhere.
- Use **Schema** view to open an interactive entity relationship diagram (ERD) of the objects in your organization. Select any object in the right sidebar to view that object's parent relationships (in blue) and child relationships (in orange). You can expand the fields of any object by clicking the + toggle button in the upper right corner of any object, and you can double-click an object to move it to the center of the view.



**Figure 6: Schema View in SoqlXplorer**



**Tip:** Simon Fell frequently adds new functionality to SoqlXplorer. To automatically check for updates, click **SoqlXplorer > Preferences** and select **Check for updates at startup**.

## See Also

[Using the Enterprise WSDL and the AJAX Toolkit to Examine Your Data Model](#) on page 18

# Choosing Between SOQL and SOSL

## Problem

You know that the platform supports Salesforce.com Object Query Language (SOQL) and Salesforce.com Object Search Language (SOSL), but you don't know what the difference is between the two, or when to use one over the other.

## Solution

A SOQL query is the equivalent of a SELECT clause in a SQL statement. Use SOQL with a `query()` call when:

- You know in which objects or fields the data resides
- You want to retrieve data from a single object or from multiple objects that are related to one another
- You want to count the number of records that meet particular criteria
- You want to sort your results as part of the query
- You want to retrieve data from number, date, or checkbox fields

A SOSL query is a programmatic way of performing a text-based search. Use SOSL with a `search()` call when:

- You don't know in which object or field the data resides and you want to find it in the most efficient way possible
- You want to retrieve multiple objects and fields efficiently, and the objects may or may not be related to one another
- You want to retrieve data for a particular division in an organization with Divisions, and you want to find it in the most efficient way possible



**Tip:** Although SOQL was previously the only one of the two query languages that allowed condition-based filtering with WHERE clauses, as of the Summer '07 release SOSL supports this functionality as well.

## See Also

- [Finding Search Data Based on Division on page 32](#)
- “Salesforce.com Object Query Language (SOQL)” at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_soql.htm)
- “Salesforce.com Object Search Language (SOSL)” at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_sosl.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_sosl.htm)

# Querying Multiple Related Objects Using Relationship Queries

## Problem

You want to use as few SOQL queries as possible to access data from multiple related objects.

## Solution

Use SOQL relationship syntax to pull data from related records in a single query.

For each of the following examples, the child object is the object on which the relationship field (the foreign key) is defined, and the parent is the object that the child references:

### Basic Child-to-Parent (Foreign Key) Traversal

To traverse a relationship from a child to a parent, use standard dot notation off the name of the relationship. For example, this SOQL query retrieves information about contacts from the Contact object, along with the name of each contact's related account (the parent object):

```
SELECT Id, LastName, FirstName, Account.Name
FROM Contact
```

Account is the name of the relationship that's defined by the AccountId lookup field on the Contact object. Using dot notation, this SOQL query retrieves the Name field on the account that is related through the Account relationship.

### Expanded Child-to-Parent (Foreign Key) Traversal

Child-to-parent traversals can extend up to five levels from the original root object. For example, the last selected field in this SOQL statement extends two levels from the root contact record by retrieving the name of the parent account on the account associated with the contact:

```
SELECT Id, LastName, FirstName, Account.Name,
       Account.Parent.Name
FROM Contact
```

### Basic Parent-to-Child (Aggregate) Traversal

To traverse a relationship from a parent to a set of children, use a nested query. For example, this SOQL query retrieves opportunities and the opportunity products associated with each opportunity:

```
SELECT Id, Name, Amount,
       (SELECT Quantity, UnitPrice, TotalPrice
        FROM OpportunityLineItems)
FROM Opportunity
```

Using the nested query, we're specifying that for each opportunity we want the respective set of OpportunityLineItem records that are related through the OpportunityLineItems child relationship.

### Combined Child-to-Parent and Parent-to-Child Traversal

Foreign key and aggregate traversals can also be combined in a single query. For example:

```
SELECT Id, Name, Account.Name,
       (SELECT Quantity, UnitPrice, TotalPrice,
```

```

        PricebookEntry.Name,
        PricebookEntry.Product2.Family
    FROM OpportunityLineItems)
FROM Opportunity

```

## See Also

- “Relationship Queries” at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql\\_relationships.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_soql_relationships.htm)
- “Salesforce.com Object Query Language (SOQL)” at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_soql.htm)

# Finding a Contact, Lead, or Person Account

## Problem

You want to write a query to find a person, but you don't know whether this person is stored as a lead, as a contact, or as a person account.

## Solution

Perform the search with one SOSL query, rather than multiple SOQL queries. For example:

- To look for *Joe* in all searchable text fields in the system, and return the IDs of the records where *Joe* is found in a case-insensitive search:

```
FIND {Joe}
```

- To look for all email fields that start with *jo* or end in *acme.com*, and return the IDs of the records where those fields are found:

```
FIND {"jo*" OR "*acme.com"}
IN EMAIL FIELDS
```



**Tip:** If you know you're looking for a name, an email address, or a phone number, it's more efficient to narrow your search scope to only name fields, email fields, or phone fields, respectively, rather than searching every field.

- To look for the name *Joe Smith* or *Joe Smythe* in the name field on a lead or contact only, and return the name and phone number of any matching record that was also created in the current fiscal quarter:

```
FIND {"Joe Smith" OR "Joe Smythe"}
IN NAME FIELDS
RETURNING
```

```
lead(name, phone WHERE createddate = THIS_FISCAL_QUARTER),
contact(name, phone WHERE createddate = THIS_FISCAL_QUARTER)
```

If you want to search for records based on a query string that was entered by a user, first escape any special characters that were entered by the user, and then construct the appropriate SOSL string. For example, the following JavaScript searches leads, contacts, and accounts for any instance of a record named “Phil Degauss”:

```
<html>
<head>
<script src="/soap/ajax/10.0/connection.js"></script>
<script type="text/javascript">
function init() {
    var who = "phil degauss";

    // These special characters must be preceded by a backslash
    // before they can be used in a SOSL query.
    who = who.replace(/([\&\|\\!\\(\()\(\)\[\]\\^~\\:\\\\+\-\-])/g, "\\\\$1");

    var sstr = "find {" + who + "} in NAME FIELDS RETURNING " +
        "Lead (id, firstname, lastname), " +
        "Contact(id, firstname, lastname), " +
        "Account(id, name)";

    // Issue the SOSL query using the AJAX Toolkit.
    var sr = sforce.connection.search(sstr);
    var m = document.getElementById('main');

    // Write out the results.
    if (sr) {
        var list = sr.getArray('searchRecords');
        for (var i = 0; i < list.length; i++ ) {
            m.innerHTML += "<p>Search results : " +
                list[i].toString();
        }
    } else {
        m.innerHTML += "<p>No search results";
    }
}
</script>
</head>

<body onload="init();">
<div id="main"></div>
</body>
</html>
```

## Discussion

You can make this solution even more robust by making use of the \* wildcard character. For example, the solution here only searches for exact matches of the name “Phil Degausse.” If you

wanted this solution to also return a record named “Philip Degasusse,” or “Phil Degasussey,” modify the user’s search string by appending \* after each token in the string:

```
var who = "phil* degauss*";
```

Note that it’s still important to maintain the space between the two names, so that each token phil\* and degauss\* will match individual name fields in the objects that are queried.

## See Also

- Choosing Between SOQL and SOSL on page 26
- “Salesforce.com Object Search Language (SOSL)” at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_sosl.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_sosl.htm)

# Retrieving Data Based on a Relative Date

## Problem

You want to retrieve records based on a relative date, such as “before last year” or “during the next fiscal quarter.”

## Solution

Use a date literal in the WHERE clause of your SOQL or SOSL statement. For example:

- This SOQL statement returns all opportunities that closed yesterday:

```
SELECT Id FROM Opportunity WHERE CloseDate = YESTERDAY
```

- This SOQL statement returns all opportunities that closed prior to the beginning of the last fiscal quarter:

```
SELECT Id FROM Opportunity WHERE CloseDate < LAST_FISCAL_QUARTER
```

- This SOQL statement returns all opportunities with a close date that is more than 15 days away:

```
SELECT Id FROM Opportunity WHERE CloseDate > NEXT_N_DAYS:15
```

## Discussion

When you specify a date in a SOQL or SOSL query, it can be a specific date or dateTime field, or it can be an expression that uses a date literal—a keyword that represents a relative range of time such as last month or next year. To construct an expression that returns date or

dateTime values within the range, use =. To construct an expression that returns date or dateTime values that fall on either side of the range, use > or <.

Salesforce.com provides date literals such as YESTERDAY, TODAY, LAST\_WEEK, NEXT\_WEEK. For a complete list, including examples and range definitions, see “Date Formats and Date Literals” in the *Force.com Web Services API Developer’s Guide* at

[www.salesforce.com/us/developer](http://www.salesforce.com/us/developer)

[/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql\\_select\\_dateformats.htm](http://docs/api/index_CSH.htm#sforce_api_calls_soql_select_dateformats.htm).

Remember that date and dateTime field values are stored as Greenwich Mean Time (GMT) or Coordinated Universal Time (UTC). When one of these values is returned in Salesforce.com, it's automatically adjusted for the time zone specified in your organization preferences.

### See Also

- “Salesforce.com Object Query Language (SOQL)” at  
[www.salesforce.com/us/developer](http://www.salesforce.com/us/developer)  
[/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql.htm](http://docs/api/index_CSH.htm#sforce_api_calls_soql.htm)
- “Salesforce.com Object Search Language (SOSL)” at  
[www.salesforce.com/us/developer](http://www.salesforce.com/us/developer)  
[/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_sosl.htm](http://docs/api/index_CSH.htm#sforce_api_calls_sosl.htm)

# Finding Search Data Based on Division

### Problem

You want to retrieve data for a particular division.

### Solution

Use the WITH clause in a SOSL query to filter on division before any other filters are applied. Although you can also filter on an object's Division field within a WHERE clause, using WITH is more efficient because it filters all records based on division before applying other filters. For example:

```
FIND {test} RETURNING Account (id where name like 'Smith'),  
      Contact (id where name like 'Smith')  
WITH DIVISION = 'Global'
```

Notice that the WITH clause filters based on the Division name field, rather than its ID. If you filter on division in a WHERE clause, you need to use the division ID instead.

## See Also

- “Salesforce.com Object Query Language (SOQL)” at  
[www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_soql.htm)
- “Salesforce.com Object Search Language (SOSL)” at  
[www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_sosl.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_sosl.htm)

# Previewing Query Results

## Problem

Your solution gives users a chance to build a query or set up a filter for a query that you've already written. You want to offer users a preview of what data is returned from their query, including the total number of records that are returned.

## Solution

Run two SOQL queries: one that uses COUNT() to return the total number of records that will be returned, and one that uses LIMIT to quickly return 25 random records that match the query.

## Discussion

If your solution allows a user to build a query or set up a filter for an existing query, there's a chance that the user might execute a long-running query that uses query() or queryMore() in a loop. This query could easily take a lot longer than the user expects.

To avoid this issue, it's a good idea to give users a preview of their query results if the result set is going to be greater than 1,000 records, including the total number of records that will be returned and a sample of what the resulting data will look like. You can then prompt them with a question such as, “Are you sure?” before proceeding with the full query.

Although running the normal query() call returns the total result size, it also returns a batch of up to 2,000 records, depending on your configured batch size. If you want your application to be faster, it's a good idea to run a COUNT() query and a LIMIT query instead.

For example, the following SOQL query returns the total number of accounts in the organization, without any filters. You can use this value in a prompt to the user to ask if they're sure they want to proceed with the query:

```
SELECT COUNT() FROM Account
```

Then you can use the following SOQL query to return a random subset of the total data to the user. The user might decide that he or she requires additional fields before the full query should run:

```
SELECT Name, BillingCity FROM Account LIMIT 25
```

### See Also

- [Implementing the `query\(\)`/`queryMore\(\)` Pattern](#) on page 238
- “Salesforce.com Object Query Language (SOQL)” at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_soql.htm)

## Sorting Query Results

### Problem

You've issued a SOQL or SOSL query and want the results sorted by the value of one or more fields.

### Solution

Use the `ORDER BY` clause in your SOQL or SOSL statement to efficiently receive results in the order that you prefer.



**Note:** You can't use the `ORDER BY` clause in any Apex query if it also uses locking. Those query results, however, are always ordered by ID.

For example, this SOQL query:

```
SELECT Name FROM Contact ORDER BY FirstName
```

Returns a list of contacts sorted alphabetically by first name:

- Andy Young
- Ashley James
- Jack Bond
- Jill Jazzy
- Stella Pavlov
- Zebidiah Jazzy

This SOQL query:

```
SELECT Name FROM Contact ORDER BY LastName DESC,
    FirstName DESC
```

Returns a list of contacts sorted in reverse-alphabetical order by last name and then in reverse-alphabetical order by first name:

- Andy Young
- Stella Pavlov
- Zebidiah Jazzy
- Jill Jazzy
- Ashley James
- Jack Bond

This SOSL query:

```
FIND {Ja*} RETURNING Contact (Name ORDER BY LastName)
```

Returns a list of contacts that include “Ja” in the name, sorted alphabetically by last name:

- Jack Bond
- Ashley James
- Jill Jazzy
- Zebidiah Jazzy

This SOSL query:

```
FIND {Ja*} RETURNING Contact (Name ORDER BY LastName,
    FirstName DESC),
    Lead (Name ORDER BY FirstName)
```

Returns a list of contacts and leads that include “Ja” in the name, where contacts are sorted alphabetically by last name and then reverse-alphabetically by first name, and where leads are sorted alphabetically by first name:

- (Contact) Jack Bond
- (Contact) Ashley James
- (Contact) Zebidiah Jazzy
- (Contact) Jill Jazzy
- (Lead) Jack Rodgers
- (Lead) Tom Jamison

### Discussion

`ORDER BY` is the best solution for sorting because the Force.com server does the work and your code doesn't need to do anything else after receiving the data.

You can sort your query results by any of the specified object's fields that is not a long text area or multi-select picklist field, even if the field is not one of the query fields that you want returned.



**Note:** If you attempt to sort by a long text area or multi-select picklist field, you'll receive a "malformed query" error message.

The `ORDER BY` clause for SOQL and SOSL includes a number of features:

- Sort by Multiple Fields

You can sort your query by multiple fields, so that records that have the same value for the first field are then ordered by the value of a second field. For example, the following query returns contacts sorted first by LastName and then by FirstName:

```
SELECT Name FROM Contact ORDER BY LastName,  
        FirstName
```

- Sort in Ascending and Descending Order

You can specify whether values should be sorted in ascending or descending order by adding the modifiers `ASC` or `DESC` to any sort field. For example, the following query returns contacts in reverse-alphabetical order:

```
SELECT Name FROM Contact ORDER BY LastName DESC,  
        FirstName DESC
```

When this value is not specified, results are sorted in ascending order by default.

- Sort Null Values

You can also specify whether null values should be sorted at the beginning (`FIRST`) or end (`LAST`) of the list of results. For example, the following query places null values at the end of a list of contact mailing cities and states that's organized by state in reverse-alphabetical order:

```
SELECT MailingCity, MailingState FROM Contact  
        ORDER BY MailingState DESC NULLS LAST
```

`ORDER BY` always follows the `WHERE` clause in a SOQL or SOSL statement. For example:

```
SELECT Name FROM Contact WHERE Name like 'Ja%'  
        ORDER BY LastName, FirstName
```



**Note:** SOQL query sorting is case insensitive. If you require case sensitive sorting, you'll need to implement this in your own code.

## See Also

- Choosing Between SOQL and SOSL on page 26
- “Salesforce.com Object Query Language (SOQL)” at  
[www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_soql.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_soql.htm)
- “Salesforce.com Object Search Language (SOSL)” at  
[www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_sosl.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_sosl.htm)

# Viewing Tags

## Problem

You want to see all the tags that are available in an organization.

## Solution

Using the API, perform the search on the TagDefinition object to retrieve multiple tags.

The following call will return a list of the public tags in alphabetic order:

```
sforce.connection.query("SELECT Name FROM TagDefinition " +  
    "WHERE Type = 'Public' ORDER BY Name");
```

An example response might be:

- Great Lakes
- Manager
- Midwest
- Northeast
- Northwest
- Senior Manager
- Southeast
- Southwest

- Staff
- Team Lead

## Discussion

Querying TagDefinition does not indicate how many times a tag is being used, nor on what type of record. To find this information, see [Viewing Records with Tags](#) on page 38.

## See Also

- [Add and Remove Tags on a Single Record](#) on page 261
- [Add and Remove Tags on Multiple Records](#) on page 263
- [Updating Tag Definitions](#) on page 265
- “TagDefinition” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/Content/sforce\\_api\\_objects\\_tagdefinition.htm](http://www.salesforce.com/us/developer/docs/api/Content/sforce_api_objects_tagdefinition.htm)

# Viewing Records with Tags

## Problem

You want to generate a list of records that use the same public tag.

## Solution

You can retrieve a list of records with a particular tag by calling queries on specific tag objects.

For instance, to retrieve a list of all contacts tagged as Staff and all contacts tagged as Great Lakes, execute the following in the AJAX Toolkit or your own client application:

```
var staffContactResults = sforce.connection.query("SELECT ItemId " +
    "FROM ContactTag WHERE Name = 'Staff'");
var greatLakesContactResults =
    sforce.connection.query("SELECT ItemId " +
    "FROM ContactTag WHERE Name = 'Great Lakes');
```

Another example: to find all contacts tagged as both Staff and Great Lakes, use the following query to form a result array with any null rows dropped:

```
var ReturnedContacts = sforce.connection.query("SELECT Name, Id, " +
    "(SELECT ItemId, Name, Id FROM Tags " +
        "WHERE Name = 'Staff' OR Name = 'Great Lakes') FROM Contact");
var TagArray = new Array();
var arraySize = 0;
for (var i = 0; i < ReturnedContacts.size; i++)
{
    if (ReturnedContacts.records[i].Tags != null)
    {
```

```

        TagArray[arraySize] = ReturnedContacts.records[i].Tags;
        arraySize++;
    }
}

```

## See Also

- [Add and Remove Tags on a Single Record](#) on page 261
- [Add and Remove Tags on Multiple Records](#) on page 263
- [Updating Tag Definitions](#) on page 265
- “TagDefinition” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_objects\\_tagdefinition.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_objects_tagdefinition.htm)
- *The AJAX Toolkit Developer’s Guide* at [www.salesforce.com/us/developer/docs/ajax/index.htm](http://www.salesforce.com/us/developer/docs/ajax/index.htm)

# Writing Shorter Queries Using Outer Joins

## Problem

You’d like to write short, simple queries similar to an outer join in SQL. For example, “retrieve all the IDs for accounts whose opportunities are all closed.”

## Solution

Use `IN` or `NOT IN` to write simple queries that exploit Salesforce.com support of semi-joins and anti-joins in SOQL.

For example, to find the account IDs for all accounts where there is a lost opportunity associated to the account, use a semi-join:

```

SELECT Id, Name
FROM Account
WHERE Id IN
(SELECT AccountId FROM Opportunity WHERE StageName = 'Closed Lost')

```

To find the account IDs for all accounts that have no open opportunities, use an anti-join query:

```

SELECT Id
FROM Account
WHERE Id NOT IN (SELECT AccountId FROM Opportunity
                  WHERE IsClosed = false
)

```

You can write nested queries using relationships. For example, to find opportunity IDs and their related line items if the line item value is greater than \$10,000, issue a query similar to the following:

```
SELECT Id, (SELECT Id from OpportunityLineItem)
FROM Opportunity
WHERE Id IN (
    SELECT OpportunityId FROM OpportunityLineItem
    WHERE totalPrice > 10000
)
```

### Discussion

Because semi-joins and anti-joins can potentially use a lot of resources during calculation, salesforce.com enforces some limits on these types of queries. For more information, see “Semi-Joins with `IN` and Anti-Joins with `NOT IN`” in the [Force.com Web Services API Developer’s Guide](#).

### See Also

- [Using the Enterprise WSDL and the AJAX Toolkit to Examine Your Data Model](#) on page 18
- [Sorting Query Results](#) on page 34

# Making Apex Work in any Organization

### Problem

You want to write Apex scripts that use standard sObjects, then you want to package your code so other organizations can download it from AppExchange. In addition, you want your code to work regardless of the standard objects available on the organization that downloads your package.

### Solution

Dynamic Apex enables developers to create more flexible applications by providing them with the ability to do the following:

- Access information about sObjects in general and the fields of an sObject.

*Describe information* for an sObject includes whether that type of sObject supports operations like create or undelete, the sObject's name and label, the sObject's fields and child objects, and so on. The describe information for a field includes whether the field has a default value, whether it is a calculated field, the type of the field, and so on.



**Note:** Describe information provides information only about *objects* in an organization, not individual records.

- Write SOQL queries, SOSL queries, and DML that are *dynamic*; that is, you don't have to know all the names, objects, or parameters when you first write the code.

*Dynamic SOQL and SOSL queries* provide the ability to execute SOQL or SOSL as a string at runtime, while *dynamic DML* provides the ability to create a record dynamically and then insert it into the database using DML. Using dynamic SOQL, SOSL, and DML, an application can be tailored precisely to the organization, as well as the user's permissions. This can be useful for applications that are installed from AppExchange.



**Note:** Because this recipe uses the development mode of Visualforce, be sure you have enabled it. To enable Visualforce development mode, click **Setup** ▶ **My personal Information** ▶ **Personal Information**, and click **Edit**. Select the Development Mode checkbox, and then click **Save**.

This recipe demonstrates using describe information for an organization. It uses a Visualforce page for displaying the information.

1. Create the Apex class that the Visualforce page uses to populate a dropdown with a list of all the sObjects available in the organization. Click **Setup** ▶ **Develop** ▶ **Apex Classes**, then click **New**.
2. Copy and paste the following into the **Body** text field for the class:

```
public class Describer {

    private Map <String, Schema.SObjectType> schemaMap =
        Schema.getGlobalDescribe();
    public List <Pair> fields {get; set;}
    public List <SelectOption> objectNames
        {public get; private set;}
    public String selectedObject {get; set;}

    // Initialize objectNames and fields
    public Describer() {
        objectNames = initObjNames();
        fields = new List<Pair>();
    }
    // Populate SelectOption list -
    // find all sObjects available in the organization
    private List<SelectOption> initObjNames() {
        List<SelectOption> objNames =
            new List<SelectOption>();
        List<String> entities =
            new List<String>(schemaMap.keySet());
        entities.sort();
        for(String name : entities)
            objNames.add(new SelectOption(name, name));
    }
}
```

```

        return objNames;
    }

    // Find the fields for the selected object
    public void showFields() {
        fields.clear();
        Map <String, Schema.SObjectField> fieldMap =
            schemaMap.get(selectedObject).getDescribe().fields.getMap();

        for(Schema.SObjectField sfield : fieldMap.Values()) {
            Schema.DescribeFieldResult dfield =
                sfield.getDescribe();
            Pair field = new Pair();
            field.key = dfield.getName();
            fields.add(field);
        }
    }

    public class Pair {
        public String key {get; set;}
        public String val {get; set;}
    }
}

```

- To create the Visualforce page, enter the following URL in your browser's address bar:

`http://MySalesforceInstance/apex/DescribePage`

Where `MySalesforceInstance` is the URL for your Salesforce.com organization. For example, if your organization uses `na3.salesforce.com`, enter `http://na3.salesforce.com/apex/DescribePage`.

- Click **Create page DescribePage** to create the page.
- At the bottom of the page, click **Page Editor**.
- Select all the text automatically generated for the page, and replace it with the following:

```

<apex:page Controller="Describer">

<apex:form id="Describe">

    <apex:pageBlock id="block2" >

        <apex:pageBlockButtons location="top" >

            <apex:commandButton
                value="Get Describe Object Fields"
                action="{!showFields}"/>

        </apex:pageBlockButtons>

```

```
<apex:pageblocksection>
    <apex:selectList value="{!!selectedObject}" size="1">
        <apex:selectOptions value="{!!objectNames}"/>
    </apex:selectList>
</apex:pageblocksection>
<apex:pageblocksection id="fieldList" rendered="{!!not(isnull(selectedObject))}">
    <apex:panelBar items="{!!fields}" var="fls">
        <apex:panelBarItem label="{!!fls.key}"/>
    </apex:panelBar>
</apex:pageblocksection>
</apex:pageBlock>
</apex:form>
</apex:page>
```

## Discussion

The controller first populates the dropdown with the list of sObjects. When a user clicks the **Get Describe Object Fields** button on the Visualforce page, the controller populates the `<apex:panelbar>` on the Visualforce page with the names of all the fields associated with that sObject.

## See Also

[Using Properties in Apex](#) on page 136

# Chapter 2

## Managing Workflow and Approvals

---

### In this chapter ...

- Managing Large Opportunities Using Time-Based Workflow
- Managing Lost Opportunities Using Workflow
- Using Workflow to Notify Case Contact for Priority Cases
- Using Workflow to Add Account Names to Opportunity Names
- Requiring Parallel Approvals for Large Campaigns
- Using a Matrix-Based Dynamic Approval Process
- Sending Outbound Messages with Workflow
- Tracking Outbound Messages from Workflow
- Updating a Field on a Parent Record

Your organization operates more efficiently with standardized internal procedures and automated business processes. Use workflow rules and approval processes to automate them. Not only do you save time, but you enforce consistency across your business practices.

Build workflow rules to trigger actions, such as email alerts, tasks, field updates, and outbound messages based on time triggers, criteria, or formulas. Use approval processes to automate all of your organization's approvals, from simple to complex.

# Managing Large Opportunities Using Time-Based Workflow

## Problem

You want to create a workflow rule to send an email alert when an opportunity with an amount greater than \$1,000,000 is created, as well as 30 days before, 15 days before, and 5 days after the opportunity close date. After the close date, you want to assign a task to the opportunity owner to update the deal.

## Solution

Create a workflow rule with multiple time triggers to send email alerts based on the criteria you specify. In this example, the rule criteria are that the opportunity must have a value greater than \$1,000,000 and must not be closed. First we'll create the workflow rule, then define the immediate and time-dependent workflow actions.

1. Create a new workflow rule.
  - a. Click **Setup** ► **Create** ► **Workflow & Approvals** ► **Workflow Rules**.
  - b. Click **New Rule**.
  - c. Select **Opportunity**.
  - d. Enter the name and description, then select **When a record is created, or when a record is edited and did not previously meet the rule criteria**.
  - e. Under **Rule Criteria**, select **Run this rule if the following** and choose **criteria are met**.
  - f. Define the rule to fire when **Opportunity: Amount greater than 1,000,000** and **Opportunity: Closed equals FALSE**.
  - g. Click **Save & Next**.
2. Add an immediate workflow action to send an email alert when an opportunity meets the above criteria.
  - a. Click **Add Workflow Action** and select **New Email**.
  - b. Define the workflow alert by providing the description, selecting the email template, and adding the recipients. In this example, the recipient might be the VP of Sales role. To find the right VP, select **Role** in the **Search** menu and click **Find**.
  - c. Click **Save**.
3. Create time triggers for large opportunities nearing their close dates, as well as one for missed opportunities.

- a. Click **Add Time Trigger** in the Time-Dependent Workflow Actions section of the Edit Workflow Rule page.
  - b. Set a time trigger for 30 days before Opportunity: Close Date and click **Save**.
  - c. Set a time trigger for 15 days before Opportunity: Close Date and click **Save**.
  - d. Set a time trigger for 5 days after Opportunity: Close Date and click **Save**.
4. Add actions to each time trigger.
    - a. Under Time-Dependent Workflow Actions, click **Add Workflow Action** and select **New Email** for each of the three time triggers. Define the alerts by providing descriptions, selecting appropriate email templates, and adding recipients.
      - For the 30 days before time trigger, the recipient might be the VP of Sales role.
      - For the 15 days before time trigger, the recipient might be the record owner.
      - For the 5 days after time trigger, the recipient might be the Executive Sponsor for the Sales team.
    - b. Click **Save** for each workflow action.
    - c. Click **Add Workflow Action** and select **New Task** for the 5 days after time trigger.
      - i. Assign the task of updating the deal to the opportunity owner.
      - ii. Set the due date to Rule Trigger Date plus 1 day.
      - iii. Select the Notify Assignee option.
      - iv. Set Status and Priority, and add any comments.
    - d. Click **Save**.
  5. Return to the Workflow Rule detail page and click **Activate** to activate the rule.

## Discussion

You want to be able to track large opportunities closely. Proactively manage your opportunities using workflow rules with multiple time triggers. As the close date approaches for a large deal, automatic workflow actions, like email alerts, increase the visibility to executives who can then take the necessary actions to close the deal. Time-dependent actions can also be used to remind management to review lost deals after the close date to determine the causes of failure, and to learn what to do next time to ensure success.

## See Also

[Managing Lost Opportunities Using Workflow](#) on page 47

# Managing Lost Opportunities Using Workflow

## Problem

You want to create a workflow rule to send an email alert when an opportunity with an amount greater than \$1,000,000 is closed and lost.

## Solution

Create a workflow rule to send email alerts based on the formula you specify. In this example, the rule criteria are that the opportunity must have a value greater than \$1,000,000 and Stage changes from Proposal/Price Quote to Closed Lost. First we'll create the workflow rule, then define the immediate workflow actions.

1. Create a new workflow rule.
  - a. Click **Setup** ► **Create** ► **Workflow & Approvals** ► **Workflow Rules**.
  - b. Click **New Rule**.
  - c. Select **Opportunity**.
  - d. Enter the name and description, then select **Every time a record is created or edited**.
  - e. Under Rule Criteria, select **Run this rule if the following** and choose **formula evaluates to true**.
  - f. Add the following formula to the formula field:

```
AND (
ISPICKVAL(StageName, "Closed Lost"),
ISPICKVAL(PRIORVALUE(StageName), "Proposal/Price Quote"),
Amount > 1000000
)
```

- g. Click **Check Syntax** to ensure there were no mistakes, then click **Save & Next**.
2. Add an immediate workflow action to send an email alert when an opportunity meets the above criteria.
  - a. Click **Add Workflow Action** and select **New Email**.
  - b. Define the workflow alert by providing the description, selecting the email template, and adding the recipients. In this example, the recipient might

be the VP of Sales role. To find the right VP, select **Role** in the Search menu and click **Find**.

- c. Click **Save**.
3. Return to the Workflow Rule detail page and click **Activate** to activate the rule.

## Discussion

If you lose a large opportunity, you want to know why. Using a workflow rule with a formula for the rule criteria, you can track specific field value changes using sophisticated formula functions, such as `ISPICKVAL` and `PRIORVALUE`. When a large opportunity is lost, automatic workflow actions, like email alerts, notify key people in your organization so that they can investigate what went wrong, and learn what to do next time to ensure success.

## See Also

[Managing Large Opportunities Using Time-Based Workflow](#) on page 45

# Using Workflow to Notify Case Contact for Priority Cases

## Problem

You want to create a workflow rule to send an email alert to the case contact when you receive a high priority case for accounts that have a Platinum service-level agreement (SLA).

## Solution

Create a workflow rule to send email alerts based on the criteria you specify. In this example, the rule criteria are that the case priority is high and the account SLA is platinum. First we'll create the workflow rule, then define the immediate workflow actions.

1. Create a new workflow rule.
  - a. Click **Setup** > **Create** > **Workflow & Approvals** > **Workflow Rules**.
  - b. Click **New Rule**.
  - c. Select **Case**.
  - d. Enter the name and description, then select **Only when a record is created**.
  - e. Under Rule Criteria, select **Run this rule if the following** and choose **criteria are met**.
  - f. Define the rule to fire when **Case: Priority equals HIGH** and **Account: SLA equals PLATINUM**.

2. Add an immediate workflow action to send an email alert to the case contact when a case meets the above criteria.
  - a. Click **Add Workflow Action** and select **New Email**.
  - b. Define the workflow alert by providing the description, selecting the email template, and adding the recipients. In this example, the recipient is the case contact.
  - c. Click **Save**.
3. Return to the Workflow Rule detail page and click **Activate** to activate the rule.

### Discussion

When a high priority account files a case, you want to know right away. Use a workflow rule with an automatic email alert to notify the right people so that they can take care of your highest priority customers. Workflow email alerts can be used to notify anyone inside or outside of your organization—they don't have to be Salesforce.com users.

## Using Workflow to Add Account Names to Opportunity Names

### Problem

You want to create a workflow rule to enforce a consistent naming standard for opportunities. The opportunity name for any opportunity that does not include the associated account name changes to *Account Name: Opportunity Name*.

### Solution

Create a workflow rule to evaluate all opportunity names based on the formula you specify. If the name does not include the associated account name, it will be added. First we'll create the workflow rule, then define the immediate actions.

1. Create a new workflow rule.
  - a. Click **Setup ▶ Create ▶ Workflow & Approvals ▶ Workflow Rules**.
  - b. Click **New Rule**.
  - c. Select the Opportunity object.
  - d. Enter the name and description, then select **When a record is created, or when a record is edited and did not previously meet the rule criteria**.
  - e. Under **Rule Criteria**, select **Run this rule if the following** and choose **formula evaluates to true**.

- f. Add the following formula to the formula field:  
`NOT (CONTAINS (Name, Account.Name))`
- g. Click **Check Syntax** to ensure there were no mistakes, then click **Save & Next**.
2. Add an immediate workflow action to update the Opportunity Name field when an opportunity meets the above criteria.
  - a. Click **Add Workflow Action** and select **New Field Update**.
  - b. Define the workflow field update by providing the name and description and selecting the Field to Update. In this example, select Opportunity Name.
  - c. Under Text Options, select Use a formula to set the new value and specify the following formula:  
`Account.Name & ":" & Name`
  - d. Click **Save**.
3. Return to the Workflow Rule detail page and click **Activate** to activate the rule.

## Discussion

You want to be able to enforce consistency across your business practices. Using a workflow rule with a formula for the rule criteria, you can set standard naming conventions for opportunities or other objects. Formulas can span multiple objects and, when used with field updates within a workflow rule, can be a powerful tool for enforcing company standards.

# Requiring Parallel Approvals for Large Campaigns

## Problem

You want to create an approval process to route all approval requests to the Director of Marketing, route all approved requests \$25,000 or greater to two designated Vice Presidents (VPs) for unanimous approval, and finally, send all requests, regardless of size, to the Director of Finance for final approval. Requests under \$25,000 that are approved by the Director of Marketing should skip the two VPs and go directly to the Director of Finance.

## Solution

Create a three-step approval process based on the criteria you specify. In this example, all requests are sent to the Director of Marketing, approved requests under \$25,000 skip the VPs

approval step and go directly to the Director of Finance; and approved requests \$25,000 or greater are sent to the VPs for parallel approval. Both must approve the request.

First we'll create the approval process, then define the approval steps and final approval/rejection actions.

1. Create a new approval process.

- a. Click **Setup > Create > Workflow & Approvals > Approval Processes**.
- b. From the **Manage Approval Processes For** drop-down list, select the object type for this approval process. For this example, select **Campaign**.
- c. Click **Create New Approval Process** and select **Use Standard Setup Wizard**.
- d. Enter the **Process Name** and **Description** for your new approval process, then click **Next**.
- e. From the **Use this approval process if the following** drop-down list, select **criteria are met**.
- f. Define the rule to fire when **Status** not equal to **Completed**, **Aborted** and **Budgeted Cost** not equal to **0**.
- g. Click **Next**.
- h. Leave the default selections for Step 3, then click **Next**.
- i. For the **Approval Assignment Email Template**, find and select the appropriate template, then click **Next**.
- j. Leave the default selections for Step 5, but select the **Display approval history information...** option, then click **Next**.
- k. By default, only the record owner—Campaign Owner, in this example—is listed under **Allowed Submitters**. Add any other users, groups, or roles that are allowed to submit requests for approval. Select the **Allow submitters to recall approval requests** option to enable the recall feature for the submitter.



**Note:** Admin users can recall approval requests, regardless of this setting. This option only enables recall for the user submitting the approval request.

- l. Click **Save**.
  - m. Select **Yes, I'd like to create an approval step now**, then click **Go**.
2. Create the first approval step.

- a. If you're not already on the **New Approval Step** page, click **New Approval Step** in the **Approval Steps** section of the Approval Process detail page.
- b. Enter the name and description. Let's call this step “Marketing Approval.” Click **Next**.

- c. For the step criteria, select All records should enter this step. Click **Next**.
  - d. Select Automatically assign to approver(s), choose **User**, and find and select the user you want. In this example, the first approver is a Director of Marketing. Since there is only one approver for this step, leave the default for the When multiple approvers are selected: setting. You can also allow the approver's delegate to approve the request.
  - e. Click **Save**.
  - f. Select the No, I'll do this later... option and click **Go**.
3. Create the second approval step.
    - a. Click **New Approval Step** in the Approval Steps section of the Approval Process detail page.
    - b. Enter the name and description. Let's call this step "Large Budget Approval." Click **Next**.
    - c. For the step criteria, select Enter this step if the following and choose **criteria are met**.
    - d. Set the criteria as Budgeted Cost greater or equal 25000.
    - e. Select Automatically assign to approver(s), choose **User**, and find and select the user you want. In this example, the first approver is the first VP.
    - f. Click **Add Row**.
    - g. Choose **User**, and find and select the user you want. In this example, the first approver is the second VP.
    - h. Select the Require UNANIMOUS approval from all selected approvers option.
    - i. Under Reject Behavior, select the Perform ONLY the rejection actions for this step and send the approval request back to the most recent approver. (Go Back 1 Step) option. If either of the VPs rejects the request, it gets sent back to the Marketing Approval step.
    - j. Click **Save**.
    - k. Select the No, I'll do this later... option and click **Go**.
  4. Create the third approval step.
    - a. Click **New Approval Step** in the Approval Steps section of the Approval Process detail page.
    - b. Enter the name and description. Let's call this step "Finance Approval." Click **Next**.
    - c. For the step criteria, select All records should enter this step.

- d. Select Automatically assign to approver(s), choose **User**, and find and select the user you want. In this example, the approver is the Director of Finance.
  - e. Under Reject Behavior, select the Perform all rejection actions for this step AND all final rejection actions. (Final Rejection) option.
  - f. Click **Save**.
  - g. Select the No, I'll do this later... option and click **Go**.
5. Go back and edit Step 2.
- a. Click **Edit** next to Step 2.
  - b. Click **Next** to go to the Specify Step Criteria page of the wizard.
  - c. For the `else` option, select **go to next step**. Approved requests less than \$25,000 are sent directly to the Finance Approval step, skipping the Large Budget Approval step.
  - d. Click **Save**.
-  **Note:** You have to create Step 3 before you can set the `else` option. This option isn't enabled for the last step of an approval process.
6. Add final approval actions for the approval process.
- a. Click **Add New**, then select **Email** in the Final Approval Actions section of the Approval Process detail page.
  - b. Add a description.
  - c. For the **Email Template**, find and select a template to use for the approval message.
  - d. For the **Search** field, select **Owner**.
  - e. Select Campaign Owner and click **Add**. On approval, the campaign owner receives email notification.
  - f. Click **Save**.
7. Add final rejection actions for the approval process.
- a. Click **Add New**, then select **Email** in the Final Rejection Actions section of the Approval Process detail page.
  - b. Add a description.
  - c. For the **Email Template**, find and select a template to use for the rejection message.
  - d. For the **Search** field, select **Owner**.
  - e. Select Campaign Owner and click **Add**. On rejection, the campaign owner receives email notification.

- f. Click **Save**.
8. Return to the Approval Process detail page and click **Activate** to activate the approval process.

## Discussion

Automating your company's campaign budget approval process enforces your best business practices. Use a single approval process to manage multiple conditions and approval requirements. Requiring multiple, unanimous approvals enforces stricter standards for large budget approvals, while the option to skip certain approval steps streamlines the process. Using parallel approvals and the "else" option, you can create sophisticated processes to match your complex business needs.

# Using a Matrix-Based Dynamic Approval Process

## Problem

You want to be able to create a single criteria-based approval process to route opportunity discount approval requests to designated approvers, based on the requester's region and the opportunity's account type.

## Solution

The Dynamic Approval Routing application, available for free on AppExchange, provides a sample solution that automates an opportunity discount approval process based on region and account type, and routes opportunities through three required levels of approval. The Dynamic Approval Routing application comes packaged with the necessary custom object, validation rule, Apex class, and Apex triggers and tests. The package also comes with documentation on how to customize the application to fit your needs.

To take advantage of dynamic approvals, do the following:

1. Go to AppExchange and install the Dynamic Approval Routing application.



**Note:** For Developer Edition organizations, or any organizations with fewer than three users, select the **Ignore Apex test failures** checkbox before installing the package. Otherwise, the Apex test fails and the installation does not succeed.

2. Once installation is complete, click All Tabs, then click **Customize My Tabs**. Add the Approval Routing Rules tab to the Selected Tabs list.
3. Customize the following fields' picklist values to match your organization.

- Owner Region—Custom picklist field on the User object to model the opportunity owner's region. Set the region for all opportunity owners.
- Account Type—Custom picklist field on the Opportunity object to model the account type associated with the opportunity. Set the account type for all opportunities.



**Note:** You may have to enable the Owner Region and Account Type fields in the Page Layouts for Users and Opportunities, respectively.

4. Create an Approval Routing Rule for each combination of region and account type.
  - a. Select the Account Type and Owner Region for the rule. The Routing Key, a composite key based on the Account Type and Owner Region fields, is created automatically.
  - b. For each rule, select approvers for Level1, Level2, and Level3, the custom user lookup fields used to model the levels of authorization. Approvers for all opportunities are assigned according to the submitter's region and the opportunity's type.
5. Create the approval process for opportunity discounts.
  - a. Click **Setup > Create > Workflow & Approvals > Approval Processes**.
  - b. From the Manage Approval Processes For drop-down list, select the object type for this approval process. For this example, select **Opportunity**.
  - c. Click **Create New Approval Process** and select **Use Standard Setup Wizard**.
  - d. Enter the Process Name (Opportunity Discount Approval) and Description for your new approval process, then click **Next**.
  - e. From the Use this approval process if the following drop-down list, select **criteria are met**.
  - f. Define the rule to fire when Closed equals False.
  - g. Click **Next**.
  - h. Leave the Next Automated Approver Determined By set at **None**, but select the Administrators OR the currently assigned approver can edit records during the approval process option, then click **Next**.
  - i. Leave the Approval Assignment Email Template blank, then click **Next**.
  - j. Select the fields you want to display on the approval page, click the Display approval history information... option, then click **Next**.
  - k. By default, only the record owner—Opportunity Owner, in this example—is listed under Allowed Submitters. Leave this setting.

1. Click **Save**.



**Note:** The approval process described here is a generic one. You can create any approval process you want to use the Approval Routing Rules you defined in the Dynamic Approval Routing custom object.

6. Create the three approval steps to correspond to the Level1, Level2, and Level3 approver fields defined in the Approval Routing Rule.
  - a. From the Approval Process Detail page, click **New Approval Step**.
  - b. Name the steps **Level 1 Approval**, **Level 2 Approval**, **Level 3 Approval**.
  - c. For the step criteria, select **All records should enter this step**.
  - d. Select **Automatically assign to approver(s)** and choose **Related User**.
  - e. Find and select **Level1**, **Level2**, and **Level3** for your three steps, respectively.
  - f. Click **Save**.

Once you become more familiar with how these routing rules work, you can design data-driven approval routing for all approval processes involving multiple routing criteria.

### Discussion

In previous Salesforce.com releases, implementing a solution for an organization with multiple account types and regions would require multiple approval processes: one for each combination of region and account type. As the numbers of account types and regions grow, the number of approval processes required jump significantly. For example, if you had two account types and three regions, you would need six approval processes; if you had 10 account types and five regions, you would need to create and maintain 50 approval processes.

By leveraging the Dynamic Approval Routing application, which uses a custom object, validation rules, and Apex code, you can create a single data-driven approval process to handle your most complex approval processes.

## Sending Outbound Messages with Workflow

### Problem

You want to send an outbound message to an external Web service when records are created or updated in Salesforce.com.

## Solution

Set up a workflow rule to send the outbound message, generate the WSDL document for the message, and then set up a listener in your language of choice.

For the following example, we'll revisit our sample Recruiting application. We'll set up a message to a legal services provider if a visa is required before a candidate can start his or her new job:

1. Click **Setup > Create > Workflow & Approvals > Workflow Rules** and create a new workflow rule that fires when a candidate is created, or when a candidate is edited and did not previously meet the rule's criteria. Set the criteria for the rule to be "Visa Required equals True."
2. Add an outbound message workflow action:
  - a. In the Immediate Workflow Actions area, click **Add Workflow Action > New Outbound Message**.
  - b. Enter a name and description for the outbound message.
  - c. Specify the `Endpoint URL` for the recipient of the message. Salesforce.com sends a SOAP message to this endpoint, which is the Web service listener that will consume the outbound message.
  - d. Select a Salesforce.com user whose security settings will control the data that's visible for the message.
  - e. Select `Include Session ID` if you want the Salesforce.com `sessionId` included in the message. You should include it if you intend to make API calls and you don't want to include a username and password in the body of your message (which is far less secure than sending the `sessionId`).
  - f. Select the field values that you want included in the outbound message.
  - g. Click **Save**.
3. Activate the workflow rule by returning to the Workflow Rule detail page and clicking **Activate**.
4. Generate the WSDL document for your outbound message: Return to the Outbound Message detail page by clicking **Setup > Create > Workflow & Approvals > Outbound Messages** and selecting the name of the outbound message. Then click **Click for WSDL**. This file is bound to the outbound message and contains the instructions about how to reach the endpoint service and what data is sent to it. Save the file to your local machine.
5. Build a listener for the outbound message. This Web service endpoint has to conform to the definition of the WSDL file. For example, to build a listener using .NET:
  - a. Run `wsdl.exe/serverInterfaceleads.wsdl` with .NET 2.0. This generates `NotificationServiceInterfaces.cs`, which defines the notification interface.
  - b. Create a class that implements `NotificationServiceInterfaces.cs`.

While there are a number of ways to do this, one simple way is to compile the interface to a .dll first (.dlls must be in the bin directory in ASP.NET):

```
mkdir bin csc /t:library /out:bin\nsi.dll
NotificationServiceInterfaces.cs
```

Then write an ASMX-based Web service that implements this interface. For example, a very simple implementation in `MyNotificationListener.asmx` might be:

```
<%@WebService class="MyNotificationListener"
    language="C#" %>
class MyNotificationListener : INotificationBinding
{
    public notificationsResponse
        notifications(notifications n)
    {
        notificationsResponse r =
            new notificationsResponse();
        r.Ack = true;
        return r;
    }
}
```

- c. Deploy the service by creating a new virtual directory in IIS for the directory that contains `MyNotificationListener.asmx`.

You can test that the service is deployed by viewing the service page with a browser. For example, if you create a virtual directory named `salesforce`, navigate to

`http://localhost/salesforce/MyNotificationListener.asmx`.

## Discussion

Although this recipe only outlines the procedure for a .NET-based solution using IIS, the process for other Web services-enabled languages and tools is similar. Note that your listener must meet the following requirements:

- It must be reachable from the public Internet.
- If it uses SSL, it must use one of the following ports:
  - 80: this port only accepts HTTP connections
  - 443: this port only accepts HTTPS connections
  - 7000-10000: these ports accept HTTP or HTTPS connections
- If it requires client certificates, you must have the current Salesforce.com client certificate available at **Setup > Develop > API**.

- The common name (CN) of the listener's certificate must match the domain name for your endpoint's server, and the certificate must be issued by a Certificate Authority trusted by the Java 2 Platform, Standard Edition 5.0 (JDK 1.5).

## See Also

- [Tracking Outbound Messages from Workflow](#) on page 59
- “Outbound Messaging” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_om\\_outboundmessaging.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_om_outboundmessaging.htm)
- The “Creating an Outbound Messaging Notification Service with CSharp and .Net Framework 2.0” whitepaper at [wiki.apexdevnet.com/index.php/Creating\\_an\\_Outbound\\_Messaging\\_Noteice\\_Service\\_with\\_CSharp\\_and\\_.Net\\_Framework\\_2.0](http://wiki.apexdevnet.com/index.php/Creating_an_Outbound_Messaging_Noteice_Service_with_CSharp_and_.Net_Framework_2.0)

# Tracking Outbound Messages from Workflow

## Problem

You want to track the status of the outbound messages that have been sent to external servers as a result of a workflow rule.

## Solution

Click **Setup > Monitoring > Outbound Messages**.

Alternatively, click **Setup > Create > Workflow & Approvals > Outbound Messages**, and then click **View Message Delivery Status**. From this page you can:

- View the status of your outbound messages, including the total number of attempted deliveries
- View the action that triggered the outbound message by clicking any workflow or approval process action ID
- Click **Retry** next to a message to immediately re-deliver the message
- Click **Del** next to a message to permanently remove the outbound message from the queue

## See Also

- [Sending Outbound Messages with Workflow](#) on page 56
- “Outbound Messaging” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_om\\_outboundmessaging.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_om_outboundmessaging.htm)

- The “Creating an Outbound Messaging Notification Service with CSharp and .Net Framework 2.0” whitepaper at [wiki.apexdevnet.com/index.php/Creating\\_an\\_Outbound\\_Messaging\\_Notification\\_Service\\_with\\_CSharp\\_and\\_.Net\\_Framework\\_2.0](http://wiki.apexdevnet.com/index.php/Creating_an_Outbound_Messaging_Notification_Service_with_CSharp_and_.Net_Framework_2.0)

## Updating a Field on a Parent Record

### Problem

You want to create a workflow rule to close an application when a candidate accepts a job offer. Applications and Candidates, both custom objects, have a master-detail relationship in the Recruiting application.

### Solution

Create a workflow rule that performs a field update on the parent record, based on the criteria you specify. In this example, when the Candidate Status field changes to “Accepted,” the Application Status field is set to “Closed.” First we’ll create the workflow rule, then define the immediate workflow actions.



**Note:** For this example, the Workflow and Roll-Up Summary Field Evaluations critical update must be activated. To do this, click **Setup > Critical Updates**. In the Updates list, find “Workflow and Roll-Up Summary Field Evaluations.” Be sure to read the Possible Impact message, then click **Activate**.

1. Create a new workflow rule:
  - a. Click **Setup > Create > Workflow & Approvals > Workflow Rules**.
  - b. Click **New Rule**.
  - c. Select the object to which this rule applies. In this example, though the field update occurs on the Application record, select the Candidate record, as that is the object that controls whether or not the rule runs. Click **Next**.
  - d. Enter a rule name and optionally, a description.
  - e. For the evaluation criteria, choose **When a record is created, or when a record is edited and did not previously meet the rule criteria**.
  - f. Under Rule Criteria, choose **criteria are met**, then specify the filter “Candidate: Status equals Accepted”.
2. Add an immediate workflow action to perform a field update when a candidate meets the criteria you specified:
  - a. Click **Edit** in the Workflow Actions section of the Workflow Rule detail page.
  - b. Click **Add Workflow Action** and select **New Field Update**.

- c. Define the field update by providing the name, description, and field to update. Since the field update occurs on the master record, select the Application object, then specify the Application: Status field.
  - d. Click **Save**.
3. Return to the Workflow Rule detail page and click **Activate** to activate the rule.

## Discussion

You want to be able to perform field updates on parent records. For standard objects, workflow rules can only perform field updates on the object related to the rule. The exceptions are that both Case Comments and Email Messages can perform cross-object field updates on Cases. For all custom objects, however, you can create workflow actions where a change to a detail record updates a field on the related master record. Cross-object field updates only work for custom-to-custom master-detail relationships.

## See Also

- [The Sample Recruiting App on page 3](#)
- “Creating Workflow Rules” in the Salesforce.com online help
- “Field Update Considerations” in the Salesforce.com online help

# Chapter 3

## Customizing the User Interface

---

### In this chapter ...

- Overriding a Standard Button
- Creating a Button with Apex
- Creating a Consistent Look and Feel with Static Resources
- Formatting a Currency
- Building a Table of Data in a Visualforce Page
- Building a Form in a Visualforce Page
- Creating a Wizard with Visualforce Pages
- Creating Custom Help
- Creating a Custom Visualforce Component
- Overriding a Standard Page
- Redirecting to a Standard Object List Page
- Dynamically Updating a Page
- Overriding a Page for Some, but not All, Users

This chapter contains recipes that help you make basic modifications to the Salesforce.com application using Apex and Visualforce.

Apex enables a new class of applications and features to be developed and deployed on the platform by providing the ability to capture business logic and rules. The language uses a combination of Java-like syntax with API functions and SOQL to let you define triggers, classes, and other representations of business logic that can interact with the platform at a low level. Conceptually similar to a stored procedure system, Apex allows almost any kind of transactional, complex logic to be developed and run entirely in the cloud.

Similar to the way Apex dramatically increases the power of developers to customize business logic, Visualforce dramatically increases the power of developers to customize the user interface. With this markup language, 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's used in standard pages, or developers can associate their own logic with a 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 user interface, while developers can work on the business logic that drives the app.

- Referencing an Existing Page
- Defining Skeleton Visualforce Templates
- Creating Tabbed Accounts
- Adding CSS to Visualforce Pages
- Editing Multiple Records Using a Visualforce List Controller
- Selecting Records with a Visualforce Custom List Controller

# Overriding a Standard Button

## Problem

You want to override what happens when a user clicks a standard button, such as **New** or **Edit**.

## Solution

To override a button on a standard object:

1. Click **Setup > Customize**, select the name of the object, and then click **Buttons and Links**.
2. In the Standard Buttons and Links related list, click **Override** next to the name of the button you want to change.
3. Select **Visualforce Page** as the content type.
4. Select a Visualforce page from the content name drop-down list.

Only Visualforce pages that use the standard controller for the object on which the button appears can be selected. 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>
```

5. Click **Save**.

To override a button on a custom object:

1. Click **Setup > Create > Objects** and select the name of the object.
2. Scroll down to the Standard Buttons and Links related list and click **Override** next to the name of the button you want to override.
3. Select **Visualforce Page** as the content type.
4. Select a Visualforce page from the content name drop-down list.

Only Visualforce pages that use the standard controller for the object on which the button appears can be selected. 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>
```

## 5. Click Save.

### Discussion

For standard and custom objects in the application, you can override the following standard buttons:

- New
- View
- Edit
- Delete
- Clone
- Accept

Additionally, some standard objects also have special actions. For example, Leads have Convert, Change Status, Add to Campaign, and others. These actions can also be overridden.

However, because Visualforce pages are only available through the Salesforce.com user interface, overriding the **New** button for contacts has no effect on new contacts that are created via Apex or the API.

You can only override standard buttons that appear on an object's detail page or list views. Buttons that only appear on an edit page or in reports can't be overridden.

As a final note, button overrides shouldn't be confused with Apex triggers, which execute in tandem with the typical behavior. Button overrides replace the standard behavior entirely. For example, if you override the **Delete** button on Accounts and a user attempts to delete an account, it won't necessarily be deleted. Instead, the user is forwarded on to the URL of your choosing, which may or may not include code to delete the account. If you define an Apex delete trigger on Accounts, however, the normal delete behavior still occurs, as long as the trigger doesn't prevent deletion by adding an error to the record.

### See Also

- [Overriding a Standard Page](#) on page 85
- [Overriding a Page for Some, but not All, Users](#) on page 93

## Creating a Button with Apex

### Problem

You want to create a new button that executes logic written in Apex.

### Solution

Define a `webService` method in Apex and then call it using the AJAX Toolkit in a button.

For example, suppose you want to create a **Mass Add Notes** button on accounts:

1. Define the Web service method in Apex by clicking **Setup > Develop > Apex Classes**, clicking **New**, and adding the following code into the body of your new class:

```
global class MassNoteInsert{

    WebService static Integer insertNotes(String iTitle,
                                         String iBody,
                                         Id[] iParentIds) {
        Note[] notes = new Note[0];
        iBody = String.escapeSingleQuotes(iBody);
        for (Id iParentId : iParentIds) {
            notes.add(new Note(parentId = iParentId,
                               title = iTitle, body = iBody));
        }
        insert notes; //Bulk Insert
        return notes.size();
    }
}
```

2. Then, click **Setup > Customize > Accounts > Buttons and Links**, and click **New** in the Custom Buttons and Links related list.
3. Name the button and assign it the following attributes:
  - Display Type: List Button
  - Behavior: Execute JavaScript
  - Content Source: OnClick JavaScript
4. In the code for the button, enter:

```
{!REQUIRESCRIPT("/soap/ajax/14.0/connection.js")}
{!REQUIRESCRIPT("/soap/ajax/14.0/apex.js")}

var idsToInsert= {!GETRECORDIDS( $ObjectType.Account )};
var noteTitle = prompt("Please enter the title of the note");
```

```

var noteBody = prompt("Please enter the body of the note");

if (idsToInsert.length) {

    // Now make a synchronous call to the Apex Web service
    // method
    var result = sforce.apex.execute(
        "MassNoteInsert",           // class
        "insertNotes",              // method
        {iTitle : noteTitle,        // method arguments
         iBody: noteBody,
         iParentIds: idsToInsert });

    alert(result[0] + " notes inserted!"); //response
} else if (idsToInsert.length == 0) {
    alert("Please select the accounts to which" +
          " you would like to add notes.");
}

```

- 5. Click Save.**
- 6. Click Setup > Customize > Accounts > Search Layouts** and add the button to the Accounts List View layout.

To test this new button, visit the Accounts tab and click **Go!** to view a list of accounts. Select one or more accounts and click **Mass Add Notes**.

## Creating a Consistent Look and Feel with Static Resources

### Problem

You want all of your Visualforce pages to have a consistent look and feel that can easily be updated.

### Solution

Create a *static resource* from a Cascading Style Sheet and include it in your Visualforce pages. A static resource is a file or archive that you upload and then reference in a Visualforce page.

To create a static resource from a stylesheet:

- 1. Click Setup > Develop > Static Resources.**
- 2. Click New Static Resource.**
- 3. In the Name text box, enter myStyleSheet. This name is used to identify the resource in Visualforce markup.**
- 4. In the Description text area, enter My site-wide stylesheet.**

5. Next to the **File** text box, click **Browse** to navigate to a local copy of your stylesheet.

6. Click **Save**.

To reference the stylesheet in a page, create the following Visualforce page:

```
<apex:page showHeader="false">
<apex:stylesheet value="{$Resource.myStyleSheet}" />
<!-- Your content goes here -->
<h1>My Style</h1>
<p>This page uses styles from my stylesheet.
The stylesheet is a static resource.</p>
</apex:page>
```

To change the look of your pages, all you need to do is update the static resource with your new styles, and every page that uses that static resource displays with your new style.

## Discussion

Static resources can be used for many other purposes, not just stylesheets. A static resource can be an archive (such as .zip and .jar files), image, stylesheet, JavaScript, or any other type of file you want to use in your Visualforce pages.

A static resource name can only contain alphanumeric characters, must begin with a letter, and must be unique in your organization. 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.

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.

## See Also

- [Creating Custom Help](#)
- “Managing Static Resources” in the Salesforce.com online help
- [Visualforce Developer’s Guide](#)

# Formatting a Currency

## Problem

You want to display a currency value, but the API doesn’t return currencies in a formatted state.

## Solution

Use Visualforce to display currencies in a manner that is correctly formatted for your users. The following Visualforce page will show Opportunity date and amount values in the format for their locale.

```
<apex:page standardController="Opportunity">
    <apex:pageBlock title="Opportunity Fields">
        <apex:pageBlockSection>
            <apex:outputField value="{!!opportunity.closedate}"/>
            <apex:outputField value="{!!opportunity.amount}"/>
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

## Discussion

The Visualforce `<apex:outputfield>` component does the formatting automatically for users in all locales.

# Building a Table of Data in a Visualforce Page

## Problem

You want to display a set of records in a table in a Visualforce page.

## Solution

Define a custom controller that returns the set of records you want to display, and then use the `<apex:dataTable>` tag to display the results. For example, assume that you want to display the contacts associated with a record in a table.

1. Create a controller so that it returns a list of associated contacts with an account record:

```
public class mySecondController {
    public Account getAccount() {
        return [select id, name,
                (select id, firstname, lastname
                 from Contacts limit 5)
                from Account where id =
                :System.currentPageReference()
                .getParameters().get('id')];
    }
}
```

2. Iterate over the resulting contacts with the `<apex:dataTable>` tag. This tag allows us to define an iteration variable that we can use to access the fields on each contact:

```
<apex:page controller="mySecondController" tabStyle="Account">

    <apex:pageBlock title="Hello {!$User.FirstName}!">
        You belong to the {!account.name} account.
    </apex:pageBlock>
    <apex:pageBlock title="Contacts">
        <apex:dataTable value="{!!account.Contacts}" var="contact" cellPadding="4" border="1">
            <apex:column>
                {!contact.FirstName}
            </apex:column>
            <apex:column>
                {!contact.LastName}
            </apex:column>
        </apex:dataTable>
    </apex:pageBlock>
</apex:page>
```

The screenshot shows a Salesforce page with the following structure:

- Hello Local!**: A page block containing the message "You belong to the Burlington Textiles Corp of America account."
- Contacts**: A page block containing an `<apex:dataTable>` component. The table has two columns: FirstName and LastName. The data is as follows:

Jack	Rogers
Leah	Cutter
Elizabeth	Rice
Mark	Leonard
Andrea	Leszek

**Figure 7: The `<apex:dataTable>` Component**

## Discussion

Notice that the `<apex:dataTable>` tag supports styling attributes like `cellPadding` and `border`. You can also style individual data elements with HTML tags. For example, the following `<apex:dataTable>` component makes the last name of each contact bold:

```
<apex:dataTable value="{!!account.Contacts}" var="contact" cellPadding="4" border="1">
    <apex:column>{!contact.FirstName}</apex:column>
    <apex:column><b>{!contact.LastName}</b></apex:column>
</apex:dataTable>
```

## See Also

- [Using Query String Parameters in a Visualforce Page](#) on page 132
- [Building a Form in a Visualforce Page](#) on page 71
- [Using AJAX in a Visualforce Page](#) on page 134
- [Creating a Wizard with Visualforce Pages](#) on page 72

# Building a Form in a Visualforce Page

## Problem

You want to create a Visualforce page that captures input from users.

## Solution

Use the `<apex:form>` tag with one or more input components and a `<apex:commandLink>` or `<apex:commandButton>` tag to submit the form.

## Discussion

To gather data for fields that are defined on a custom or standard object, use the `<apex:inputField>` tag. This tag renders the appropriate input widget based on the 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.

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

```
<apex:page standardController="Account">
<apex:form>
    <apex:pageBlock title="Hello {!$User.FirstName}!">
        You belong to the {!account.name} account.<p/>
        Account Name:
        <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.

Notice that the `<apex:commandButton>` tag is associated with the `save` action of the standard controller, which performs the same action as the **Save** button on the standard edit page. The `<apex:inputField>` tag is bound to the account name field by setting the `value` attribute with an expression containing dot notation.

The screenshot shows a Visualforce page with the title "Hello Local!". It contains a message: "You belong to the Burlington Textiles Corp of America account." Below this is an "Account Name:" label followed by an input field containing "Burlington Textiles Corp". At the bottom is a "Save New Account Name" button.

**Figure 8: The `<apex:form>` Component with a Single Input Field**

The `<apex:inputField>` tag can be used with either standard or custom controllers and enforces all security restrictions and other flags on the field, such as whether a value for the field is required, or whether it must be unique from the value on all other records of that type. Its only drawback is that if it's used to display variables in a custom controller that aren't bound to an object field, the variables might not display the way you want them to.

To gather data for these variables, use the `<apex:inputCheckbox>`, `<apex:inputHidden>`, `<apex:inputSecret>`, `<apex:inputText>`, or `<apex:inputTextarea>` tags instead. To learn more about these tags, browse the component library by clicking **Component Reference** in the Page Editor or accessing it through the Salesforce.com online help.

## See Also

- [Using Query String Parameters in a Visualforce Page](#) on page 132
- [Building a Table of Data in a Visualforce Page](#) on page 69
- [Using AJAX in a Visualforce Page](#) on page 134
- [Creating a Wizard with Visualforce Pages](#) on page 72

# Creating a Wizard with Visualforce Pages

## Problem

You want to create a three-step opportunity 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

## Solution

Define three Visualforce 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.

It's important to understand the best procedure for creating the Visualforce pages, since the three pages and the custom controller reference each other. In what appears to be a Catch-22, you can't create the controller without the pages, but the pages have to exist in order for you to refer to them in the controller.

Luckily, we can work our way out of this conundrum by defining a page that's completely empty. To create the wizard pages and controller:

1. Navigate to the URL for the first page:  
<https://<host>.salesforce.com/apex/opptyStep1>
2. Click **Create Page newOpptyStep1**.
3. Repeat the two steps above for the other pages in the wizard: opptyStep2 and opptyStep3.



**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.

Now all three of the pages exist. Even though they are empty, they need to exist before we can create a controller that refers to them.

4. 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 following controller code:

```
/*
 * This class is the controller behind the New Customer Opportunity
 * wizard. The new wizard is comprised of three pages, each of
 * which utilizes the same instance of this controller.
 */
public class newOpportunityController {

    // These four class 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 class
    // 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 are used to control navigation through
// the wizard. Each returns a reference to one of the three pages
// in the wizard.
public PageReference step1() {
    return Page.newOpptyStep1;
}

public PageReference step2() {
    return Page.newOpptyStep2;
}

public PageReference step3() {
    return Page.newOpptyStep3;
}

// 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.
        // Note that using '/' in the new PageReference object keeps
        // the user in the current instance of salesforce, rather than
        // redirecting him or her elsewhere.
        PageReference opptyPage = new PageReference('/' +
                                         opportunity.id);
        opptyPage.setRedirect(true);

        return opptyPage;
    }

}

```

**5.** Save the controller.

**6.** Navigate to the URL for the first page:

<https://<host>.salesforce.com/apex/opptyStep1> and copy in the following:

```

<apex:page controller="newOpportunityController"
           tabStyle="Opportunity">
    <apex:sectionHeader title="New Customer Opportunity"
                        subtitle="Step 1 of 3"/>
    <apex:form>
        <apex:pageBlock title="Customer Information">

            <!-- This facet tag defines the "Next" button that appears
                in the footer of the pageBlock. It calls the step2()
                controller method, which returns a pageReference to
                the next step of the wizard. -->
            <apex:facet name="footer">
                <apex:commandButton action="{!step2}" value="Next"
                                    styleClass="btn"/>
            </apex:facet>
            <apex:pageBlockSection title="Account Information">

                <!-- <apex:panelGrid> tags organize data in the same way as
                    a table. It places all child elements in successive cells,
                    in left-to-right, top-to-bottom order -->
                <!-- <apex:outputLabel> and <apex:inputField> tags can be
                    bound together with the for and id attribute values,
                    respectively. -->
                <apex:panelGrid columns="2">
                    <apex:outputLabel value="Account Name" for="accountName"/>
                    <apex:inputField id="accountName" value="{!!account.name}"/>
                </apex:panelGrid>
            </apex:pageBlockSection>
        </apex:pageBlock>
    </apex:form>
</apex:page>

```

```

<apex:outputLabel value="Account Site" for="accountSite"/>
<apex:inputField id="accountSite" value="{!account.site}"/>
</apex:panelGrid>
</apex:pageBlockSection>
<apex:pageBlockSection title="Contact Information">
<apex:panelGrid columns="2">
    <apex:outputLabel value="First Name"
        for="contactFirstName"/>
    <apex:inputField id="contactFirstName"
        value="{!contact.firstName}"/>
    <apex:outputLabel value="Last Name" for="contactLastName"/>
    <apex:inputField id="contactLastName"
        value="{!contact.lastName}"/>
    <apex:outputLabel value="Phone" for="contactPhone"/>
    <apex:inputField id="contactPhone"
        value="{!contact.phone}"/>
</apex:panelGrid>
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>

```

**7.** Save the page.

**8.** Navigate to the URL for the first page:

<https://<host>.salesforce.com/apex/opptyStep2> and copy in the following:

```

<apex:page controller="newOpportunityController"
    tabStyle="Opportunity">
    <apex:sectionHeader title="New Customer Opportunity"
        subtitle="Step 2 of 3"/>
    <apex:form>
        <apex:pageBlock title="Opportunity Information">
            <apex:facet name="footer">
                <apex:outputPanel>
                    <apex:commandButton action="{!step1}" value="Previous"
                        styleClass="btn"/>
                    <apex:commandButton action="{!step3}" value="Next"
                        styleClass="btn"/>
                </apex:outputPanel>
            </apex:facet>
            <apex:pageBlockSection title="Opportunity Information">
                <apex:panelGrid columns="2">
                    <apex:outputLabel value="Opportunity Name"
                        for="opportunityName"/>
                    <apex:inputField id="opportunityName"
                        value="{!opportunity.name}"/>
                    <apex:outputLabel value="Amount"
                        for="opportunityAmount"/>
                    <apex:inputField id="opportunityAmount"
                        value="{!opportunity.amount}"/>
                    <apex:outputLabel value="Close Date"
                        for="opportunityCloseDate"/>
                    <apex:inputField id="opportunityCloseDate"

```

```

                value=" {!opportunity.closeDate}"/>
<apex:outputLabel value="Stage"
                   for="opportunityStageName"/>
<apex:inputField id="opportunityStageName"
                  value=" {!opportunity.stageName}"/>
<apex:outputLabel value="Role for Contact:
                    {!contact.firstName}
                    {!contact.lastName}"
                   for="contactRole"/>
<apex:inputField id="contactRole"
                  value=" {!role.role}"/>
</apex:panelGrid>
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>
```

**9.** Save the page.

**10.** Navigate to the URL for the first page:

<https://<host>.salesforce.com/apex/opptyStep3> and copy in the following:

```

<apex:page controller="newOpportunityController"
           tabStyle="Opportunity">
<apex:sectionHeader title="New Customer Opportunity"
                     subtitle="Step 3 of 3"/>
<apex:form>
<apex:pageBlock title="Confirmation">
<apex:facet name="footer">
<apex:outputPanel>
<apex:commandButton action="!step2" value="Previous"
                     styleClass="btn"/>
<apex:commandButton action="!save" value="Save"
                     styleClass="btn"/>
</apex:outputPanel>
</apex:facet>
<apex:pageBlockSection title="Account Information">
<apex:panelGrid columns="2">
<apex:outputText value="Account Name"/>
<apex:outputText value=" {!account.name}"/>
<apex:outputText value="Account Site"/>
<apex:outputText value=" {!account.site}"/>
</apex:panelGrid>
</apex:pageBlockSection>
<apex:pageBlockSection title="Contact Information">
<apex:panelGrid columns="2">
<apex:outputText value="First Name"/>
<apex:outputText value=" {!contact.firstName}"/>
<apex:outputText value="Last Name"/>
<apex:outputText value=" {!contact.lastName}"/>
<apex:outputText value="Phone"/>
<apex:outputText value=" {!contact.phone}"/>
<apex:outputText value="Role"/>
<apex:outputText value=" {!role.role}"/>
```

```
</apex:panelGrid>
</apex:pageBlockSection>
<apex:pageBlockSection title="Opportunity Information">
    <apex:panelGrid columns="2">
        <apex:outputText value="Opportunity Name"/>
        <apex:outputText value="{!opportunity.name}"/>
        <apex:outputText value="Amount"/>
        <apex:outputText value="{!opportunity.amount}"/>
        <apex:outputText value="Close Date"/>
        <apex:outputText value="{!opportunity.closeDate}"/>
    </apex:panelGrid>
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>
```

### 11. Save the page.

You'll now have a wizard that is composed of the following three pages:

The screenshot shows a wizard step titled "New Customer Opportunity Step 1 of 3". The "Customer Information" section is expanded, showing the "Account Information" and "Contact Information" sections. The "Account Information" section contains fields for "Account Name" and "Account Site". The "Contact Information" section contains fields for "First Name", "Last Name", and "Phone". A "Next" button is located at the bottom of the page.

**Figure 9: Step 1 of the New Customer Opportunity Wizard**



**New Customer Opportunity**  
Step 2 of 3

**Opportunity Information** [Previous](#) [Next](#) [Cancel](#)

Opportunity Information		<small>[ ] = Required Information</small>	
Opportunity Name	<input type="text"/>	Amount	<input type="text"/>
Close Date	<input type="text"/> [ 2/24/2009 ]	Stage	<input type="text"/> --None-- <a href="#">▼</a>
Role	<input type="text"/> --None-- <a href="#">▼</a>		

[Previous](#) [Next](#) [Cancel](#)

Figure 10: Step 2 of the New Customer Opportunity Wizard



**New Customer Opportunity**  
Step 3 of 3

**Confirmation** [Previous](#) [Save](#) [Cancel](#)

**▼ Account Information**

Account Name	Andrew's Party Supply, Inc	Account Site	San Francisco
--------------	----------------------------	--------------	---------------

**▼ Contact Information**

First Name	Andrew	Last Name	Waite
Phone	(415) 555-1234	Role	Decision Maker

**▼ Opportunity Information**

Opportunity Name	5000 Kazoos	Amount	\$399.00
Close Date	2/24/2009		

[Previous](#) [Save](#) [Cancel](#)

Figure 11: Step 3 of the New Customer Opportunity Wizard

You can start the wizard by navigating to the first page:  
<https://<host>.salesforce.com/apex/opptyStep1>.

## Discussion

On page 1 of the wizard, data about the associated contact and account is gathered from the user. Notice the following about the code for the first page of the wizard:

- Some tags, including `<apex:pageBlock>`, can take an optional `<apex:facet>` child element that controls the header or footer of the component. The order in which the facet tag appears in the `<apex:pageBlock>` body doesn't matter because it includes a name attribute that identifies where the element should be placed. In this page of the wizard, the facet tag defines the **Next** button that appears in the footer of the `pageBlock` area.
- An `<apex:commandButton>` tag represents a user control that executes a method in the controller class. 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. Command buttons must appear in a form, because the form component itself is responsible for refreshing the page display based on the new `pageReference`.

```
<apex:facet name="footer">
    <apex:commandButton action="{!step2}" value="Next"
        styleClass="btn"/>
</apex:facet>
```

- An `<apex:panelGrid>` tag organizes a set of data for display. Similar to a table, it simply takes a number of columns and then places all child elements in successive cells, in left-to-right, top-to-bottom order. For example, in the Account Information area of this page, the "Account Name" label is in the first cell, the input field for `Account Name` is in the second cell, the "Account Site" label is in the third cell, and the input field for `Account Site` is in the fourth.

```
<apex:panelGrid columns="2">
    <apex:outputLabel value="Account Name" for="accountName"/>
    <apex:inputField id="accountName" value="{!account.name}"/>
    <apex:outputLabel value="Account Site" for="accountSite"/>
    <apex:inputField id="accountSite" value="{!account.site}"/>
</apex:panelGrid>
```

- The `value` attribute on the first `<apex:inputField>` tag in the preceding code excerpt assigns the user's input to the `name` field of the account record that's returned by the `getAccount()` method in the controller.
- The `<apex:outputLabel>` and `<apex:inputField>` tags can be bound together when the `id` attribute value on `<apex:inputField>` tag matches the `for` attribute value on `<apex:outputLabel>`. Binding these tags together improves the user experience because it provides special behavior in the Web page. For example, clicking on the label puts the cursor in the associated input field. Likewise, if the input is a checkbox, it toggles the checkmark.

On page 2 of the wizard, data about the opportunity is gathered from the user. Notice the following about the code for the second page of the wizard:

- Because this page displays two buttons in the `pageBlock` footer, they're wrapped in an `<apex:outputPanel>` tag. This tag needs to be used because `<apex:facet>` expects only one child component.



**Note:** You also must use an `<apex:panelGroup>` tag within an `<apex:panelGrid>` if you want to place more than one component into a single cell of the grid.

- Although the code 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.

On page 3 of the wizard, the user can choose to save the changes or cancel. Notice that the third page of the wizard simply writes text to the page with `<apex:outputText>` tags.

## See Also

- Using Query String Parameters in a Visualforce Page on page 132
- Building a Table of Data in a Visualforce Page on page 69
- Building a Form in a Visualforce Page on page 71
- Using AJAX in a Visualforce Page on page 134

# Creating Custom Help

## Problem

You've extended Salesforce.com and developed new functionality for your users. Now you need to tell them how to use this new functionality.

## Solution

You can use the `<apex:outputLink>` 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:

- Upload the PDF as a static resource named `customhelp`.
- On your Visualforce page, create the following link:

```
<apex:outputLink
    value="{!URLFOR($Resource.customhelp)}"
    target="_blank">Help for this page
</apex:outputLink>
```

Notice that the static resource reference is wrapped in a `URLFOR` function, which allows the page to redirect properly.

You can take advantage of other options on the `<apex:outputLink>` component, such as the `target` attribute, to control how the help is displayed.

## Discussion

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 link:

```
<apex:outputLink  
    value="{!URLFOR ($Resource.customhelpsystem)  
        /index.htm"}>Help for this page  
</apex:outputLink>
```

When a user clicks the help link, the `index.htm` file in the static resource displays. To change the content of your online help, simply update the static resource (ZIP file), and every page that uses that static resource displays with your changes.

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.

## See Also

- [Creating a Consistent Look and Feel with Static Resources](#) on page 67
- “Managing Static Resources” in the Salesforce.com online help
- “Managing Custom Objects” in the Salesforce.com online help
- [Visualforce Developer’s Guide](#)

# Creating a Custom Visualforce Component

## Problem

You want to use the same functionality on multiple Visualforce pages, but there isn't a standard component to do it.

## Solution

You can create a custom component and use that on your Visualforce page. The following custom component allows your users to easily increase or decrease a value on the page. First, create the component, and then add it to a Visualforce page.

To create a custom component:

1. In Salesforce.com click **Setup > Develop > Components**.
2. Click **New**.
3. In the Label text box, enter Add or Subtract Values.
4. In the Name text box, enter addSubValue.
5. In the Description text box, enter Increase or decrease a value.
6. In the Body text box, enter the following Visualforce markup:

```
<apex:component>

<!-- Attribute Definitions -->
<apex:attribute name="myvalue"
    description="Default value for the component."
    type="Integer" required="true"/>
<apex:attribute name="max"
    description="Maximum value"
    type="Integer" required="true"/>
<apex:attribute name="min"
    description="Minimum value"
    type="Integer" required="true"/>

<!-- JavaScript definitions -->
<script>
    function increment(valueId) {
        if(document.getElementById(valueId).value < {!max})
        {
            document.getElementById(valueId).value++;
        }
        else
        {
            alert("You can't increase the number above " + {!max});
        }
    }

    function decrement(valueId) {
        if(document.getElementById(valueId).value > {!min})
        {
            document.getElementById(valueId).value--;
        }
        else
        {
            alert("You can't decrease the number below " + {!min});
        }
    }
</script>
```

```
<!-- Custom Component Definition -->
<table cellspacing='0' cellpadding='0'>
<tr>
  <td rowspan="2">
    <apex:inputText value="{!!myvalue}" size="4" id="theValue"/>
  </td>
  <td>
    <div onclick="increment('{$Component.theValue}')";>&#9650;</div>
  </td>
</tr>
<tr>
  <td>
    <div onclick="decrement('{$Component.theValue}')";>&#9660;</div>
  </td>
</tr>
</table>
</apex:component>
```

## 7. Click **Save**.

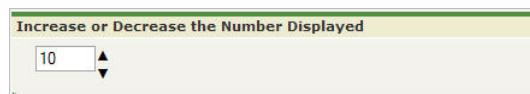
To add your new component to a page:

1. Click **Setup****Develop** ▶ **Pages**.
2. Click **New**.
3. In the name field, enter `testAddSub`.
4. Optionally enter a label and description.
5. In the editor, enter the following markup:

```
<apex:page>
  <apex:pageBlock title="Increase or Decrease the Number Displayed">
    <apex:form>
      <c:addSubValue myvalue="10" min="0" max="15"/>
    </apex:form>
  </apex:pageBlock>
</apex:page>
```

## 6. Click **Save**.

When you open the page in a browser, you will see something like the following:



Click the up or down arrow to increase or decrease the number displayed. If you try to change it to a value lower than 0 or greater than 15, an error message displays in a popup dialog.

## Discussion

This solution illustrates how you can mix Visualforce, HTML, and JavaScript within a custom component. This combination allows you to create very powerful custom components. To really make this component useful, you can bind the value to a field (like the number of employees on an account) and update that field when your user submits the form.

The name of a custom component can only contain alphanumeric characters, start with a letter, and must be unique from all other components in your organization.

The description of a custom component appears in the component reference with other standard component descriptions. If your custom component includes `apex:attribute` components, the description will be included in the component reference as well.

A single component can hold up to 1 MB of text, or approximately 1,000,000 characters.

When working on a custom component, you can click **Quick Save** to save your changes and continue editing your component.

The Visualforce markup must be valid before you can save your component.

## Overriding a Standard Page

### Problem

You want to override what happens when a user clicks a tab in Salesforce.com, such as the Account or Contact tab.

### Solution

To override a standard object tab:

1. Create a Visualforce page called `accountOverride`.
  - a. Click **Setup** ▶ **Develop** ▶ **Pages**.
  - b. Click **New**.
  - c. In the label field, enter **Override the Account Home Page**.
  - d. In the name field, enter `accountOverride`.
  - e. In the description, enter **This page will display for all users**.
  - f. In the editor, enter the following markup:

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

```
<apex:form>
    <apex:pageBlock>
        <apex:pageBlockTable value="={!accounts}" var="a">
            <apex:column value=" {!a.name} "/>
            <apex:column value=" {!a.lastmodifieddate} "/>
            <apex:column value=" {!a.owner.alias} "/>
        </apex:pageBlockTable>
    </apex:pageBlock>
    <apex:panelGrid columns="2">
        <apex:commandLink action=" {!previous}">
            Previous
        </apex:commandlink>
        <apex:commandLink action=" {!next}">
            Next
        </apex:commandlink>
    </apex:panelGrid>
</apex:form>
</apex:page>
```

- a. Click **Save**.
2. Set the page level security to allow all users to view the page.
  - a. Click **Security** for the page you just created.
  - b. Select all the profiles in the Available Profiles list and click **Add** to add them to the Enabled Profiles list.
  - c. Click **Save**.
3. Create an override that directs users to your new page.
  - a. Click **Setup** ▶ **Customize** ▶ **Accounts** ▶ **Buttons and Links**.
  - b. In the Standard Buttons and Links list, click **Override** for the Accounts Tab.
  - c. Set the content type to **Visualforce Page**.
  - d. From the **Content Name** drop-down list, select **accountOverride**.
  - e. Click **Save**.

## Discussion

Overriding a tab overrides what a user sees when he or she clicks on the tab. Overriding a standard button overrides its functionality in all parts of the Salesforce.com user interface. For example, if you override a **New** button for contacts, it overrides the **New** button on the Contacts tab, the **New** button on any Contacts related list, and the **Contact** option in the Create New drop-down list in the sidebar.

You can override the home tab for all standard and custom objects.

## See Also

- Overriding a Standard Button on page 64
- Dynamically Updating a Page on page 88

# Redirecting to a Standard Object List Page

## Problem

You want to redirect a user to the standard account list page.

## Solution

For buttons or links that navigate a user to the accounts tab, redirect them to a Visualforce page with the following content:

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

The user will see a page that resembles the following:

Action	Account Name	Billing State/Province	Phone	Type
<a href="#">Edit</a>   <a href="#">Del</a>	Ace Textiles Corp of America	NC	(336) 555-1212	Customer
<a href="#">Edit</a>   <a href="#">Del</a>	Dickerson PLC	KS	(785) 555-3252	Customer
<a href="#">Edit</a>   <a href="#">Del</a>	Edwards Communications	TX	(512) 555-6396	Customer
<a href="#">Edit</a>   <a href="#">Del</a>	Express Logistics	OR	(503) 555-0745	Partner
<a href="#">Edit</a>   <a href="#">Del</a>	Global Media	Ontario	(905) 555-1212	Prospect
<a href="#">Edit</a>   <a href="#">Del</a>	Graham Inc	CA	(650) 555-5267	Customer
<a href="#">Edit</a>   <a href="#">Del</a>	Piper Hotels and Resorts	IL	(312) 555-4092	Customer

**Figure 12: Overriding the Account Detail Page**

## Discussion

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)}"/>
```

## See Also

- Overriding a Standard Button on page 64
- Overriding a Standard Page on page 85
- Overriding a Page for Some, but not All, Users on page 93

## Dynamically Updating a Page

### Problem

You have fields on a page in your Salesforce.com application that you only want displayed when the user has made a specific selection. In this example, you want to capture specific information about a lost opportunity. During the normal progression of a deal, there is no reason to display a field called `Reason_Lost`. You want this field to display only when the Stage moves to `Closed_Lost`. In addition to the `Reason_Lost` field, you would like a required lookup field that contains competitors names, so that the user can specify which competitor won the deal.

### Solution

Using Visualforce, you can create a page that updates just a portion of that page based on a changed value.

Because you're merely extending the existing behavior of the Force.com platform, you don't need to create a controller (an Apex class).

Some Visualforce components are AJAX aware and allow you to add AJAX behaviors to a page without having to write any JavaScript. 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 a Visualforce component.

This example creates two new fields for Opportunity: `Primary_Compétitor` and `Reason_Lost`. After you create the fields, you need to create a Visualforce page that uses these fields with an existing opportunity.

To create the new fields:

1. In Salesforce.com click **Setup** > **Customize** > **Opportunities** > **Fields**.
2. In Opportunity Custom Fields & Relationships, click **New**.
3. Select **Lookup Relationship** and click **Next**.
4. For **Related To** select **Account** and click **Next**.
5. In the **Field Label** text box, enter `Primary_Compétitor`.
6. In the **Field Name** text box, enter `Primary_Compétitor`.
7. Add a brief description in the **Description** text box, and what should display as help text in the **Help Text** text box as a best practice.
8. Click **Next**.
9. In Enterprise, Unlimited, and Developer Editions, accept the default field-level security and click **Next**.

10. Use the default values to make the field appear on all page layouts, and click **Next**.
11. Use the default value for the related list label, and accept the defaults for all the page layouts where the related list appears.
12. Click **Save and New** so you can immediately create the second field.
13. Select **Text Area (Long)** and click **Next**.
14. In the **Field Label** text box, enter **Reason Lost**.
15. In the **Field Name** text box, enter **Reason\_Lost**.
16. Add a brief description in the **Description** text box, and what should display as help text in the **Help Text** text box as a best practice.
17. Click **Next**.
18. Use the default values and click **Next** for field-level security, adding the reference to page layouts, and adding custom related lists.
19. Click **Save**.

To create the Visualforce page:

1. In Salesforce.com click **Setup > Develop > Pages**.
2. Click **New**.
3. In the **Label** text box, enter **Dynamic Opportunity Edit**.
4. In the **Name** text box, enter **opportunityEdit**.
5. In the **Description** text box, enter **Dynamically edit an opportunity**.
6. In the **Body** text box, delete the existing Visualforce markup and enter the following instead:

```
<apex:page standardController="Opportunity" sidebar="false">
<apex:sectionHeader title="Edit Opportunity"
                     subtitle="{!!opportunity.name}"/>
<apex:form>
    <apex:pageBlock title="Edit Opportunity" id="thePageBlock"
                   mode="edit">
        <apex:pageMessages />
        <apex:pageBlockButtons>
            <apex:commandButton value="Save" action="{!!save}"/>
            <apex:commandButton value="Cancel" action="{!!cancel}"/>
        </apex:pageBlockButtons>
        <apex:actionRegion>
            <apex:pageBlockSection title="Basic Information"
                                   columns="1">
                <apex:inputField value="{!!opportunity.name}"/>
                <apex:pageBlockSectionItem>
                    <apex:outputLabel value="Stage"/>
                    <apex:outputPanel>
                        <apex:inputField value="{!!opportunity.stageName}">
                            <apex:actionSupport event="onchange"
                                               rerender="thePageBlock"
                                               status="status"/>
                        </apex:inputField>
                        <apex:actionStatus startText="applying value..." id="status"/>
                    </apex:outputPanel>
                </apex:pageBlockSectionItem>
            </apex:pageBlockSection>
        </apex:actionRegion>
    </apex:pageBlock>
</apex:form>
```

```
</apex:outputPanel>
</apex:pageBlockSectionItem>
<apex:inputField value="{!!opportunity.amount}" />
<apex:inputField value="{!!opportunity.closedate}" />
</apex:pageBlockSection>
</apex:actionRegion>
<apex:pageBlockSection title="Closed Lost Information"
    columns="1"
    rendered="{!!opportunity.stageName == 'Closed Lost'}">
    <apex:inputField
        value="{!!opportunity.Primary_Competitor__c}"
        required="true"/>
    <apex:inputField value="{!!opportunity.Reason_Lost__c}" />
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:page>
```

## 7. Click Save.

To test the Visualforce page:

### 1. Find the ID for an opportunity record:

- a. Click the Opportunities tab and find an opportunity that hasn't been closed.
- b. Click the name of that opportunity. In the URL, the ID of the opportunity record follows the name of the instance of Salesforce.com your organization uses. For example, in the following URL,  
<https://na3.salesforce.com/006D000000C4V1N>,  
006D000000C4V1N is the ID of an opportunity record.

### 2. Call the page by using the following URL:

```
https://salesforce_instance/apex/opportunityEdit?id=ID
```

Substitute the instance of Salesforce.com that you use, such as na3.salesforce.com, for salesforce\_instance. Substitute the opportunity record ID you found in the previous step for ID. For example,  
<https://na3.salesforce.com/apex/opportunityEdit?id=006D000000C4V1D>.

Your page should resemble the following:

**Edit Opportunity**

**GenePoint Lab Generators**

**Basic Information**

Opportunity Name	GenePoint Lab Generators
Stage	Closed Lost
Amount	60,000.00
Close Date	9/2/2006 [ 9/11/2008 ]

**Closed Lost Information**

Primary Competitor	[Search]
Reason Lost	[Search]

**Save** **Cancel**

**Figure 13: Edit Opportunity Page**

## Discussion

Note that the Basic Information portion of the page (the top page block) is contained in an `<apex:actionRegion>` tag:

```
<apex:actionRegion>
    <apex:inputField value="{!opportunity.name}" />
    .
    .
    <apex:inputField value="{!opportunity.stageName}" />
    .
    .
    <apex:inputField value="{!opportunity.amount}" />
    .
    .
    <apex:inputField value="{!opportunity.closedate}" />
    .
    .
</apex:actionRegion>
```

The `<apex:actionRegion>` tag encapsulates which components should be processed by the Force.com server when the AJAX partial page update request to rerender the page is made. Because these fields are in an action region, they do not get passed to the server until the user

clicks **Save**: all the data is still in memory. This is important when you have field dependencies. For example, once the user specifies Closed Lost, the `Primary Competitor` field becomes required. If the updated field information was sent back to the server before the user could update the field, you would receive an error about the required field.

When a user changes the value of `Stage` to Closed Lost, a message displays stating “applying value...” This message is generated by the following markup in the Visualforce page.

```
<apex:inputField value=" {!opportunity.stageName}">
    <apex:actionSupport event="onchange"
        rerender="thePageBlock"
        status="status"/>
</apex:inputField>

<apex:actionStatus startText="applying value..." id="status"/>
```

The input field is `opportunity.stageName`, the event is `onchange`, and the status is `status`. This is bound to the `<apex:actionStatus>` tag by the ID `status`.

In addition to controlling the message that displays, this input field also controls the rendering of the page. If the field changes, this field specifies that a component named `thePageBlock` should be rerendered. The entire page is contained in an `<apex:pageBlock>` tag with the ID `thePageBlock`:

```
<apex:pageBlock title="Edit Opportunity" id="thePageBlock" mode="edit">
```

Toward the bottom of the Visualforce page markup is an `<apex:pageBlockSection>` section. This section is only rendered when the `opportunity.stageName` field has been set to Closed Lost:

```
<apex:pageBlockSection title="Closed Lost Information"
    columns="1"
    rendered=" {!opportunity.stageName == 'Closed Lost'}">
    <apex:inputField value=" {!opportunity.Primary_Competitor__c}"
        required="true"/>
    <apex:inputField value=" {!opportunity.Reason_Lost__c}"/>
</apex:pageBlockSection>
```

## See Also

- [Using AJAX in a Visualforce Page](#) on page 134
- [Overriding a Page for Some, but not All, Users](#) on page 93

# Overriding a Page for Some, but not All, Users

## Problem

Some of your users should use a custom Visualforce page, while others should use a standard Salesforce.com page.

## Solution

To override the Account tab with a Visualforce page for most of your users, but send users with the “System Administrator” profile to the standard Salesforce.com Account home page:

1. Create a Visualforce page called conditionalAccountOverride.
  - a. Click **Setup** > **Develop** > **Pages**.
  - b. Click **New**.
  - c. In the label field, enter **Override the Account Page for Most Users**.
  - d. In the name field, enter **accountOverride**.
  - e. In the description, enter **This page will display for all users, except System Administrators, when they click the Account tab.**
  - f. In the editor, enter the following markup:

```
<apex:page action=
    "{!if($Profile.Name !='System Administrator',
        null,
        urlFor($Action.Account.Tab, $ObjectType.Account,
        null, true))}"
    standardController="Account"
    recordSetVar="accounts"
    tabStyle="Account">

    <!-- Replace with your markup -->
    This page replaces your Account home page for
    all users except Administrators.
</apex:page>
```

- g. Click **Save**.
2. Set the page level security to allow all users to view the page.
  - a. Click **Security** for the page you just created.
  - b. Select all the profiles that will be using this page in the Available Profiles list and click **Add** to add them to the Enabled Profiles list.
  - c. Click **Save**.

3. Create an override that directs users to your new page.
  - a. Click **Setup > Customize > Accounts > Buttons and Links.**
  - b. In the Standard Buttons and Links list, click **Override** for the Accounts Tab.
  - c. Set the content type to **Visualforce Page**.
  - d. From the **Content Name** drop-down list, select **conditionalAccountOverride**.
  - e. Click **Save**.

## Discussion

This solution uses the `action` attribute on the `apex:page` component to test the user's profile. Using the `action` attribute is a good way to ensure an action is taken when the page loads.

If instead of limiting the standard page to a particular group of users, you want to limit the override to a particular group of users, such as all “Marketing Users” and “Solution Managers,” you need to create a controller extension.

To override the Account tab with a custom Visualforce page only for users with the “Marketing User” or the “Solution Manager” profile:

1. Create a Visualforce page called `standardAcctPage`.
  - a. Click **Setup > Develop > Pages**.
  - b. Click **New**.
  - c. In the label field, enter **Override the Account Page for Most Users**.
  - d. In the name field, enter `standardAcctPage`.
  - e. In the description, enter **This page will display for all users, except Marketing Users and Solution Managers, when they click the Account tab.**
  - f. In the editor, enter the following markup:

```
<apex:page standardController="account"
           extensions="overrideCon"
           action="{!!redirect}">
    <apex:detail/>
</apex:page>
```

This page overrides your current Account home page. It uses a controller extension to test the user profile. If the user is a “Marketing User” or “Solution Manager,” they are redirected to a different page.

2. Using the procedure in step 1, create a second Visualforce page called customAcctPage:

```
<apex:page standardController="account">
    <h1>Override Account page for two profiles</h1>
    <apex:detail />
</apex:page>
```

This is the page that only the “Marketing Users” and “Solutions Managers” will see. They only get to this page through redirection.

3. Grant access to both pages for all profiles.
  - a. Click **Security** for each of the pages you just created.
  - b. Select all the profiles that will be using this page in the Available Profiles list and click **Add** to add them to the Enabled Profiles list.
  - c. Click **Save**.
4. Create a controller extension called overrideCon.
  - a. Click **Setup ▶ Develop ▶ Apex Classes**.
  - b. Click **New**.
  - c. In the editor, add the following content:

```
public class overrideCon {
    String recordId;

    public overrideCon(ApexPages.StandardController
        controller) {recordId = controller.getId();}

    public PageReference redirect() {
        Profile p = [select name from Profile where id =
            :UserInfo.getProfileId()];
        if ('Marketing User'.equals(p.name)
            || 'Solution Manager'.equals(p.name) )
        {
            PageReference customPage =
Page.customAccountPage;
            customPage.setRedirect(true);
            customPage.getParameters().put('id', recordId);
            return customPage;
        } else {
            return null; //otherwise stay on the same page
        }
    }
}
```

- d. Click **Save**.
5. Create an override that directs users to the standardAcctPage page when they click on the Accounts tab.

- a. Click **Setup > Customize > Accounts > Buttons and Links**.
- b. In the Standard Buttons and Links list, click **Override** for the Accounts Tab.
- c. Set the content type to **Visualforce Page**.
- d. From the **Content Name** drop-down list, select **standardAcctPage**.
- e. Click **Save**.

When a user clicks on the Account tab, if their profile is “Marketing User” or “Solution Manager” the controller extension will automatically redirect them to the `customAcctPage`.

### See Also

- For more information about page level security and creating home page overrides, see the Salesforce.com online help.
- For more information about creating page overrides, see [Overriding a Standard Button](#) on page 64.
- For more information about creating controller extensions, see [Custom Controllers and Controller Extensions](#) in the Visualforce Developer Guide.

## Referencing an Existing Page

### Problem

You want to create a Visualforce page that is used in exactly the same way in multiple places.

### Solution

Use the `<apex:include>` tag 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.

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 `templateExample` 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>

```

Nothing happens if you click **Save**. This is the 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.

## Discussion

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>
```

Although the displayed text changes, the `templateExample` logic remains the same.

## See Also

- [Creating a Custom Visualforce Component](#) on page 82
- [Defining Skeleton Visualforce Templates](#) on page 98

# Defining Skeleton Visualforce Templates

## Problem

You want to create a Visualforce page that looks the same in multiple places, but its fields and values change with each implementation.

## Solution

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:include>`, 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>
```

## Discussion

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 doesn't 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 don't 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.

## See Also

- [Creating a Consistent Look and Feel with Static Resources](#) on page 67
- [Referencing an Existing Page](#) on page 96

# Creating Tabbed Accounts

## Problem

In Salesforce.com, 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. You'd like a different way to display the information for accounts.

## Solution

Using a Visualforce page, you can make each section for an account display in a tab, such as contacts, opportunities, and so on.

To create a tabbed view for account, first you have to create a Visualforce page. Then you have to override the standard account view to use the page.

To create the Visualforce page:

1. In Salesforce.com click **Setup ▶ Develop ▶ Pages**.
2. Click **New**.
3. In the **Label** text box, enter **Tabbed Account View**.
4. In the **Name** text box, enter **tabbedAccount**.
5. In the **Description** text box, enter **Displays account information in tabs**.
6. In the **Body** text box, delete the existing Visualforce markup and enter the following instead:

```
<apex:page standardController="Account" showHeader="true"
           tabStyle="account" >
    <apex:tabPanel switchType="client"
                  selectedTab="tabdetails"
                  id="AccountTabPanel">
        <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>
```

## 7. Click Save.

To override the standard account view:

1. Click **Setup** ▶ **Customize** ▶ **Accounts** ▶ **Buttons and Links**.
2. In the Standard Buttons and Links related list, click **Override** next to View.
3. For **Content Type** select Visualforce Page.
4. For **Content Name** select tabbedAccount.
5. Click **Save**.

## Discussion

To see your changes, select the Account tab, then select an account to view. Across the top of the page you should see a list of tabs: Details, Contacts, Opportunities, Open Activities and Notes and Attachments.

The tabbed display looks like the following:

The screenshot shows the Salesforce Account Detail page for the account "Edge Communications". At the top, there is a navigation bar with tabs: Accounts, Contacts, Opportunities, Forecasts, Contracts, Cases, Solutions, Products, Reports, and Document. Below the navigation bar, the account name "Edge Communications" is displayed next to a folder icon. On the right side of the header, there are links for "Printable View" and "Customize". The main content area has a tabbed interface. The "Details" tab is currently selected and highlighted in green. Other tabs include "Contacts", "Opportunities", "Open Activities", and "Notes and Attachments". Below the tabs, there is a section titled "Account Detail" with various fields. Some fields have change links (e.g., "Anthony Di [Change]"), while others are plain text. The fields include: Account Owner (Anthony Di), Account Name (Edge Communications), Parent Account (None), Account Number (CD451796), Account Site (None), Type (Customer - Direct), Industry (Electronics), Annual Revenue (\$139,000,000), Rating (Hot), Phone ((512) 757-6000), Fax ((512) 757-9000), Website (<http://edgecomm.com>), Ticker Symbol (EDGE), Ownership (Public), Employees (1,000), and SIC Code (6576). There is also a "Hello" button at the bottom of the detail section.

**Figure 14: Account Detail Page Displayed as Tabbed View**

## See Also

- Overriding a Standard Page on page 85
- Overriding a Standard Button on page 64
- Adding CSS to Visualforce Pages on page 104

# Adding CSS to Visualforce Pages

## Problem

You want to set your pages apart by changing color, font, or other display options on the page.

## Solution

Add CSS markup to your page, then have the appropriate component refer to it.

This example expands [Creating Tabbed Accounts](#) on page 102.

The following Visualforce page markup is from [Creating Tabbed Accounts](#) on page 102. It creates tabs to display account information.

```
<apex:page standardController="Account" showHeader="true"
           tabStyle="account">
    <apex:tabPanel switchType="client" selectedTab="tabdetails">
```

```

        id="AccountTabPanel">
<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>

```

One thing you may notice about this page is that all of the tabs display in the same color. When you select a tab, it does not change color. Adding the following CSS markup to the Visualforce page causes the color of the active tab to change when the user selects it.

To add CSS markup to the page:

1. In Salesforce.com click **Setup ▶ Develop ▶ Pages**.
2. Click **Edit** next to the page `tabbedAccount`.
3. Add this code immediately following the `<apex:page>` tag. This Visualforce page markup defines the colors for the active tab and inactive tabs, as CSS:

```

<style>
    .activeTab {background-color: #236FBD; color:white;
                background-image:none}
    .inactiveTab {background-color: lightgrey; color:black;
                  background-image:none}
</style>

```

4. Replace the existing `<apex:tabPanel>` markup with the following:

```

<apex:tabPanel switchType="client" selectedTab="tabdetails"
    id="AccountTabPanel" tabClass="activeTab"
    inactiveTabClass="inactiveTab">

```

This code sets values for the following:

- `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.

## 5. Click Save.

Here is the complete updated Visualforce page markup:

```
<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>
```

The tabbed display with colored tabs looks like the following:

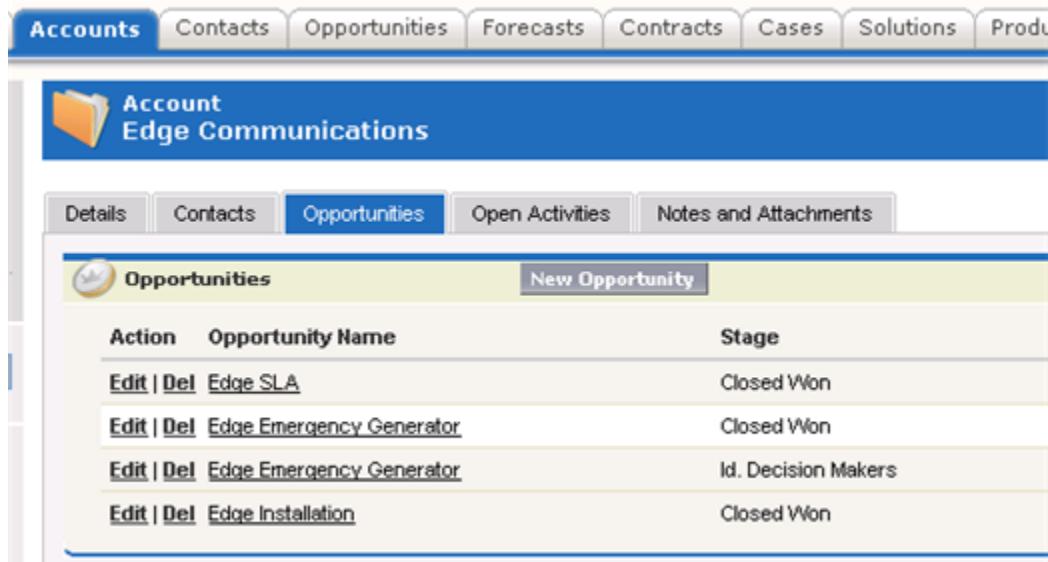


Figure 15: Account Detail Page Displayed as Tabbed View

### See Also

- [Creating Tabbed Accounts](#) on page 102
- [Overriding a Standard Page](#) on page 85

## Editing Multiple Records Using a Visualforce List Controller

### Problem

You need to edit a set of records at the same time. A standard detail page, though, only allows you to edit one record at a time.

### Solution

Create a Visualforce page using a *standard list controller*. The standard list controller enables you to create Visualforce pages that can display or act on a set of records.

To create a Visualforce page using a standard list controller to edit a list of opportunities:

1. In Salesforce.com click **Setup > Develop > Pages**.
2. Click **New**.

3. In the Label text box, enter Multiple opportunity edit.
4. In the Name text box, enter multOppEdit.
5. In the Description text box, enter Edit multiple opportunity records in a set.
6. In the Body text box, delete the existing Visualforce markup and enter the following instead:

```
<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>
```

## 7. Click **Save**.

Call the page by using the following URL:

[https://salesforce\\_instance/apex/multOppEdit](https://salesforce_instance/apex/multOppEdit)

Substitute *salesforce\_instance* for the instance of Salesforce.com that you use, such as na3.salesforce.com. For example,

<https://na3.salesforce.com/apex/multOppEdit>.

Your page should resemble the following:

Opportunity Name	Stage	Close Date
Burlington Textiles Weaving Plant Generator	Closed Won	9/2/2006 [10/7/2008]
Dickenson Mobile Generators	Qualification	9/2/2006 [10/7/2008]
Edge Emergency Generator	Closed Won	9/2/2006 [10/7/2008]
Edge Emergency Generator	Id. Decision Makers	9/2/2006 [10/7/2008]
Edge Installation	Closed Won	9/2/2006 [10/7/2008]
Edge SLA	Closed Won	9/2/2006 [10/7/2008]
Express Logistics Portable Truck Generators	Value Proposition	9/2/2006 [10/7/2008]
Express Logistics SLA	Closed Lost	9/2/2006 [10/7/2008]

**Figure 16: Editing Multiple Opportunities on a Single Page**

## Discussion

Using a standard list controller is very similar to using a standard controller. First you set the `standardController` attribute on the `<apex:page>` component. This specifies the type of records that you want to access. Then set the `recordSetVar` attribute on the same component. This indicates that you're using a standard set controller.

```
<apex:page standardController="Opportunity"
           recordSetVar="opportunities"
           tabStyle="Opportunity"
           sidebar="false">
```

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

```
<apex:pageBlockTable value="{!opportunities}" var="opp">
```

In this example, two fields, `Stage` and `Close Date` are displayed for edit in the table. They each form a column in the table:

```
<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>
```

## See Also

- [Building a Table of Data in a Visualforce Page](#) on page 69

- [Creating a Wizard with Visualforce Pages](#) on page 72

## Selecting Records with a Visualforce Custom List Controller

### Problem

You need to edit a set of records, but you want to exclude irrelevant ones.

### Solution

Create a Visualforce page using a custom list controller. The custom list controller can define which records to present using SOQL.

### Discussion

A custom list controller is similar to a standard list controller, except it implements Apex logic to define actions on a set of records.

You can also create a custom list controller that uses anti- and semi-joins as part of the SOQL query. Anti-joins exclude records that match certain criteria, while semi-joins includes records.

The following custom list controller uses an anti-join to retrieve all accounts that don't have any open opportunities. It is implemented as a controller extension:

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

    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 standard list controller actions (`{!previous}` and `{!next}`). However, the set of available records depends on the list 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>
```

## See Also

- [Editing Multiple Records Using a Visualforce List Controller](#) on page 110
- [Overriding a Page for Some, but not All, Users](#) on page 93
- [Retrieving Data Based on a Relative Date](#) on page 31

# Chapter 4

## Displaying Data and Modifying Data Actions

---

### In this chapter ...

- Creating a Many-to-Many Relationship
- Storing and Displaying Confidential Information
- Averaging Aggregated Data
- Displaying Fields from a Related Record on a Detail Page
- Blocking Record Creation with Cross-Object Validation Rules
- Validating Data Based on Fields in Other Records
- Using Query String Parameters in a Visualforce Page
- Using AJAX in a Visualforce Page
- Using Properties in Apex
- Mass Updating Contacts When an Account Changes
- Bulk Processing Records in a Trigger

After using recipes in previous chapters that help you modify the look and feel of Salesforce.com or inspect existing data, you may wish to modify how specific data is presented, or change the behavior of data actions. Use the recipes in this chapter to explore the Force.com platform: specify how confidential information is displayed, the default behavior of bulk processing of records, the handling of duplicate records, and many other data display or data action behaviors.

- Using Batch Apex to Reassign Account Owners
- Controlling Recursive Triggers
- Comparing Queries Against Trigger.old and Trigger.new
- Preventing Duplicate Records from Saving
- Creating a Child Record When a Parent Record is Created
- Using Custom Settings to Display Data
- Using System.runAs in Test Methods
- Integrating Visualforce and Google Charts
- Using Special Characters in Custom Links

## Creating a Many-to-Many Relationship

### Problem

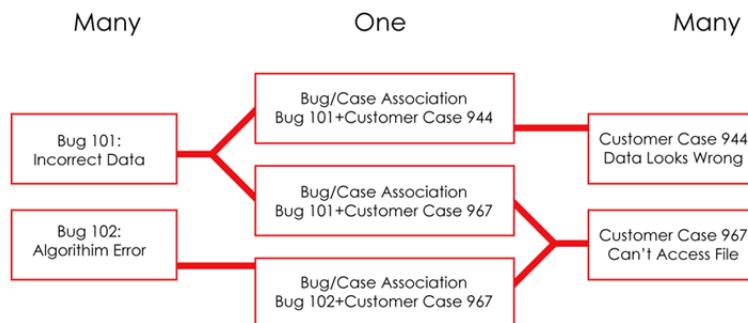
You want to model a many-to-many relationship between objects in which each record of one object can be related to many records of the other object, and vice versa. For example, a customer case can require many bug fixes, and a bug fix can resolve multiple customer cases.



**Note:** This recipe has been provided by salesforce.com Training & Certification and is drawn from the expert-led training courses available around the world. Salesforce.com training courses provide an opportunity to get hands-on experience with the Force.com platform and Salesforce.com applications as well as to prepare you to become Salesforce.com Certified. Register for a course at [www.salesforce.com/training](http://www.salesforce.com/training).

### Solution

Relate the two objects using a *custom junction object*, and customize the junction object related lists and reports. A custom junction object is an object with two master-detail relationships. Its purpose is to create an association between two other objects. For example, a many-to-many relationship between bugs and cases uses a custom junction object called BugCaseAssociation to associate the Bug and Case objects.



**Figure 17: A Many-to-Many Relationship Between Bugs and Customer Cases**

To create the junction object:

1. Click **Setup** ▶ **Create** ▶ **Objects**.
2. Click **New Custom Object**.

3. In the custom object wizard, consider these tips specifically for junction objects:

- Name the object with a label that indicates its purpose, such as `BugCaseAssociation`.
- For the `Record Name` field, use the auto-number data type.
- Do not launch the custom tab wizard before clicking **Save**. Junction objects do not need a tab.

To create the two master-detail relationships:

1. Verify that the two objects you want to relate to each other already exist. For example, you may want to relate the standard case object to a custom bug object.
2. On the junction object, create the first master-detail relationship field. In the custom field wizard:
  - a. Choose `Master-Detail Relationship` as the field type.
  - b. Select one of the objects to relate to your junction object. For example, select `Case`.

The first master-detail relationship you create on your junction object becomes the *primary* relationship. This affects the following for the junction object records:

- Look and feel: The junction object's detail and edit pages use the color and any associated icon of the primary master object.
  - Record ownership: The junction object records inherit the value of the `Owner` field from their associated primary master record. Because objects on the detail side of a relationship do not have a visible `Owner` field, this is only relevant if you later delete both master-detail relationships on your junction object.
  - Division: If your organization uses divisions to segment data, the junction object records inherit their division from their associated primary master record. Similar to the record ownership, this is only relevant if you later delete both master-detail relationships.
- c. Select a `Sharing Setting` option. For master-detail relationship fields, the `Sharing Setting` attribute determines the sharing access that users must have to a master record in order to create, edit, or delete its associated detail records.
  - d. For the `Related List Label` that will display on the page layout of the master object, do not accept the default. Change this to use the name of the other master object in your many-to-many relationship. For example, change this to `Bugs` so users will see a `Bugs` related list on the case detail page.

3. On the junction object, create the second master-detail relationship. In the custom field wizard:

- a. Choose Master-Detail Relationship as the field type.
- b. Select the other desired master object to relate to your junction object. For example, select Bug.

The second master-detail relationship you create on your junction object becomes the *secondary* relationship. If you delete the primary master-detail relationship or convert it to a lookup relationship, the secondary master object becomes primary.

- c. Select a Sharing Setting option. For master-detail relationship fields, the Sharing Setting attribute determines the sharing access that users must have to a master record in order to create, edit, or delete its associated detail records.
- d. For the Related List Label that will display on the page layout of the master object, do not accept the default. Change this to use the name of the other master object in your many-to-many relationship. For example, change this to Cases so users will see a Cases related list on the bug detail page.

To customize the fields that display in the junction object related list on each master object page layout:

1. Edit the page layout of each master object that is related to the junction object. For example, to modify the BugCaseAssociations related list for case records, edit the page layout for cases at **Setup > Customize > Cases > Page Layouts**.
2. Edit the properties of the related list you want to modify. For example, on cases the BugCaseAssociations related list was renamed to Bugs, so select the Bugs related list.
3. Add the fields to display in the related list. You can add fields from the junction object itself, but more importantly, you can add fields from the other master object.

Each field is prefixed with its object name in the popup window. In the related list itself, only fields from the junction object are prefixed with the object name; fields from the other master object are not.



**Note:** The junction object related list does not include an icon on the master record's detail pages because the junction object does not have a custom tab. If you make a tab for the junction object, the icon is included.

## Discussion

For a many-to-many relationship in Salesforce.com, each master object record displays a related list of the associated junction object records. To create a seamless user experience, you can change the name of the junction object related list on each of the master object page layouts to have the name of the other master object. For example, you might change the BugCaseAssociations related list to Cases on the bugs page layout and to Bugs on the cases page layout. You can further customize these related lists to display fields from the other master object.

Review the following considerations before creating many-to-many relationships between objects:

- Junction object records are deleted when either associated master record is deleted. If both associated master records are deleted, the junction object record is deleted permanently and cannot be restored.
- Sharing access to a junction object record is determined by a user's sharing access to both associated master records and the *Sharing Setting* option on the relationship field. For example, if the sharing setting on both parents is Read/Write, then the user must have Read/Write access to *both* parents in order to have Read/Write access to the junction object.
- In a many-to-many relationship, a user cannot delete a parent record if there are more than 200 junction object records associated with it *and* if the junction object has a roll-up summary field that rolls up to the other parent. To delete this object, manually delete junction object records until the count is fewer than 200.
- Roll-up summary fields that summarize data from the junction object can be created on both master objects.
- Formula fields and validation rules on the junction object can reference fields on both master objects.
- You can define Apex triggers on both master objects and the junction object.
- A junction object cannot be on the master side of another master-detail relationship.
- Junction objects cannot be on the subscriber side of a Salesforce to Salesforce connection.
- You cannot create a many-to-many self relationship, that is, the two master-detail relationships on the junction object cannot have the same master object.

Many-to-many relationships provide two standard report types that join the master objects and the junction object. The report types are:

- “Primary master with junction object and secondary master” in the primary master object's report category
- “Secondary master with junction object and primary master” in the secondary master object's report category

The order of the master objects in the report type is important. The master object listed first determines the scope of records that can be displayed in the report.

You can create custom reports based on these standard report types. In addition, you can create custom report types to customize which related objects are joined in the report.

### See Also

- “Considerations for Relationships” in the Salesforce.com online help
- “Customizing Page Layouts” in the Salesforce.com online help
- “What is a Custom Report Type?” in the Salesforce.com online help

## Storing and Displaying Confidential Information

### Problem

You want to store employee Social Security numbers as encrypted data as required by government regulations or industry standards. Only select certain users should be able to view the entire social security number; all other users should only be able to view the last four digits. In addition, you want to ensure that users enter the numbers in the standard social security number format, including the dashes after the third and fifth digits.

### Solution

On the standard user object, create an encrypted custom field to store the user's Social Security number. Set the field's Mask Type attribute to hide the first five digits of the social security number, and add field-level help to inform users of the required format. Then, create a validation rule that uses the REGEX () function to verify that the value of the custom field is in the correct format. Finally, create a new custom profile that allows a select group of users to see the Social Security numbers unmasked.



**Note:** To enable encrypted fields for your organization, contact salesforce.com Customer Support.

1. Define the encrypted custom field.
  - a. Click **Setup** ▶ **Customize** ▶ **Users** ▶ **Fields**.
  - b. In the User Custom Fields related list, click **New**.
  - c. Select **Text (Encrypted)**, and click **Next**.
  - d. In the **Field Label** field, enter **Social Security Number**.
  - e. In the **Length** field, enter **11**. This allows the field to accept all nine digits of the Social Security number plus the dashes after the third and fifth digits.
  - f. In the **Description** field, enter **Encrypted Social Security Number field**.
  - g. In the **Help Text** field, enter information to help your users understand what value to type. For example, Enter your Social Security

- number. Remember to include dashes after the third and fifth digits.
- h.** In the Mask Type field, select Social Security Number. This option hides the first five digits (it hides the first 7 characters) and displays the last four. Only users with profiles that have the “View Encrypted Data” permission selected are able to view all nine digits of the Social Security number.
  - i.** In the Mask Character field, select the character, either an asterisk (\*) or an X, to use for hidden characters.
  - j.** Click **Next**.
  - k.** In Enterprise, Unlimited, and Developer Editions, set the field-level security to determine whether the field should be visible or read only for specific profiles. These settings determine whether or not the field itself is visible, but do not affect whether or not the user sees the masked or full Social Security number. You will specify the type of masking when you create the custom profile.
  - l.** Click **Next**.
  - m.** Leave the Add Field and User Layout checkboxes selected.
  - n.** Click **Save**.
- 2.** Create the validation rule.
- a.** Click **Setup > Customize > Users > Validation Rules**.
  - b.** Click **New**.
  - c.** In the Rule Name field, enter Social Security Number Format Check.
  - d.** In the Description field, enter Validates that the Social Security Number is in the correct format.
  - e.** Enter the following error condition formula:
- ```
NOT (
OR (
LEN (Social_Security_Number__c) = 0,
REGEX( Social_Security_Number__c ,
"[0-9]{3}-[0-9]{2}-[0-9]{4}")
)
)
```
- f.** Click **Check Syntax** to make sure the syntax is correct.
  - g.** In the Error Message field, enter a message that appears if the user enters a Social Security number in an invalid format. For example, the message might read: The Social Security number you entered is not in the correct format. The correct format is 999-99-9999.
  - h.** In the Error Location field, specify whether you want the error message you entered above to appear at the top of the page or next to the field. If

you choose **Field**, select the Social Security Number field in the adjacent drop-down list.

- i. Click **Save**.
3. Create the custom profile.
  - a. Click **Setup > Manage Users > Profiles**.
  - b. Click **New**.
  - c. Select an existing profile to copy.
  - d. Name the new custom profile.
  - e. Click **Save**.
  - f. Click **Edit**.
  - g. In the General User Permissions section, select the **View Encrypted Data** checkbox. This allows users with this profile to see the complete value of encrypted fields instead of the masking characters.
  - h. Click **Save**.
4. Assign the new custom profile to the users allowed to view the encrypted data.

### Discussion

Government regulations and industry standards require many companies to use encryption to protect their most sensitive employee and customer data. Encrypted custom fields can help companies comply with these regulations. Salesforce.com encrypts these fields with 128-bit keys and uses the AES (Advanced Encryption Standard) algorithm which has been adopted as an encryption standard by the U.S. government. Encrypted custom fields should only be used when regulations require encryption because they involve additional processing and have search-related limitations.

To further protect the confidentiality of encrypted custom field values, Salesforce.com requires you to specify a mask type for each encrypted field you create. Character masking lets you hide the characters in encrypted field values, allowing users to see the full value of an encrypted custom field only if their profile has the “View Encrypted Data” permission. If your company uses parts of confidential data, such as the last four digits of a person’s Social Security or credit card number, to verify the identity of customers, configure your encrypted custom fields to use a mask type that reveals only those digits, such as the **Last Four Characters Clear** mask type.

In addition to ensuring your data's confidentiality, you also want to ensure its accuracy. Validation rules improve the quality of your data by verifying that the data a user enters in a record meets the standards you specify before the user can save the record. A validation rule contains a formula expression that evaluates the data in one or more fields and returns a value of “True” or “False.” If the validation rule returns “True,” Salesforce.com lets the user save the record; otherwise, Salesforce.com displays an error message.

The validation rule in this recipe uses the `REGEX()` function, which compares the custom field to a regular expression. A regular expression is a string used to describe a format of a string according to certain syntax rules. Salesforce.com regular expression syntax is based on [Java Platform SE 6 syntax](#); however, backslash characters (\) must be changed to double backslashes (\\\) because backslash is an escape character in Salesforce.com.

## See Also

- [Validating Data Based on Fields in Other Records](#) on page 129
- “About Validation Rules” in the Salesforce.com online help
- “Operators and Functions” in the Salesforce.com online help
- “About Encrypted Custom Fields” in the Salesforce.com online help

# Averaging Aggregated Data

## Problem

You want to calculate the average value of a numeric field on a set of detail records in a master-detail relationship.



**Note:** This recipe has been provided by salesforce.com Training & Certification and is drawn from the expert-led training courses available around the world. Salesforce.com training courses provide an opportunity to get hands-on experience with the Force.com platform and Salesforce.com applications as well as to prepare you to become Salesforce.com Certified. Register for a course at [www.salesforce.com/training](http://www.salesforce.com/training).

## Solution

Create two roll-up summary fields: one that sums a numeric field on a detail record and another that counts the number of detail records. Then use a formula field that divides the first roll-up summary field by the second.

To illustrate this example, we'll look at the Job Application and Review objects in the sample Recruiting application. The Job Application object is the master in a master-detail relationship with the Review object. The Review object has a 1-5 rating system. We want to display the average rating on the job application.

To create the first roll-up summary field:

1. Click **Setup** ▶ **Create** ▶ **Objects**.
2. Click **Job Application**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the **Roll-Up Summary** data type, and click **Next**.

5. In the Field Label field, enter Total Rating. Once you move your cursor, the Field Name text box should be automatically populated with Total\_Rating.
6. Click **Next**.
7. In the Summarized Object drop-down list, choose Reviews.
8. Under Select Roll-Up Type, select SUM.
9. In the Field to Aggregate drop-down list, select **Rating**.
10. Leave All records should be included in the calculation selected, and click **Next**.
11. Configure the remaining field-level security and page layout settings as desired.
12. Click **Save**.

To create the second roll-up summary field:

1. Click **Setup > Create > Objects**.
2. Click **Job Application**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the Roll-Up Summary data type, and click **Next**.
5. In the Field Label field, enter Number of Reviews. Once you move your cursor, the Field Name text box should be automatically populated with Number\_of\_Reviews.
6. Click **Next**.
7. In the Summarized Object drop-down list, choose Reviews.
8. Under Select Roll-Up Type, select COUNT.
9. Leave All records should be included in the calculation selected, and click **Next**.
10. Configure the remaining field-level security and page layout settings as desired.
11. Click **Save**.

To create the formula field:

1. Click **Setup > Create > Objects**.
2. Click **Job Application**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the Formula data type, and click **Next**.
5. In the Field Label field, enter Average Rating. Once you move your cursor, the Field Name text box should be automatically populated with Average\_Rating.
6. Select the Number formula return type and click **Next**.
7. Enter the following formula:  

```
IF(Number_of_Reviews__c > 0, Total_Rating__c / Number_of_Reviews__c, 0)
```
8. Click **Next**.
9. Configure the remaining field-level security and page layout settings as desired.

## 10. Click Save.

### Discussion

Roll-up summary fields let you easily display values from a set of detail records. Use roll-up summary fields to display the total number of detail records, the sum of all the values in a detail record field, or the highest or lowest value of a detail field.

Before you begin working with roll-up summary fields, note that they are only available on the master object in a master-detail relationship.

When working with merge fields in formulas, use the `IF` function to ensure that the formula field displays correctly even if they have an invalid value. (For example, an invalid value may occur if the formula divides by zero.) The `IF` function in this recipe ensures that the formula displays a value when there are one or more reviews, and displays a zero if there are no reviews; otherwise, the formula field might display `#Error`.

### See Also

- [Blocking Record Creation with Cross-Object Validation Rules](#) on page 126
- “About Roll-Up Summary Fields” in the Salesforce.com online help
- “Examples of Advanced Formula Fields” in the Salesforce.com online help
- “Considerations for Relationships” in the Salesforce.com online help

## Displaying Fields from a Related Record on a Detail Page

### Problem

You want to show field values from a related object on a detail page.

### Solution

Use a cross-object formula field to retrieve and display the field values from a related object.

To illustrate this example, we'll look at the Review object in the sample Recruiting application. The Review object is the detail record of the Job Application object. The Job Application object has lookup relationships to the Position and Candidate objects. Using cross-object formulas, we will display the title of the related position and the name of the related candidate on each review record.

1. Click **Setup** ▶ **Create** ▶ **Objects**.
2. Click **Review**.

3. In the Custom Fields & Relationships related list, click **New**.
4. Select the **Formula** data type, and click **Next**.
5. In the **Field Label** field, enter **Position**. Once you move your cursor, the **Field Name** text box should be automatically populated with **Position**.
6. Select the **Text** formula return type and click **Next**.
7. Click the Advanced Formula tab.



**Note:** You can create cross-object formulas only on the Advanced Formula tab.

8. Click the **Insert Field** button.
9. Select **Review >** in the first column. The second column displays all of the Review object's fields as well as its related objects, which are denoted by a greater-than (>) sign .
10. Select **Job Application >** in the second column. The third column displays the fields of the Job Application object.
11. Select **Position>** in the third column. The fourth column displays the fields of the Position object.

Be sure that you select **Position >** (with the greater-than sign) and not **Position**. The one with the greater-than sign is the Position object, while the one without the greater-than sign is the Position lookup field on the Job Application object.

12. Choose **Position Title** in the fourth column.
13. Click **Insert**.

Your formula now looks like this:

```
Job_Application__r.Position__r.Name
```

14. Click **Next**.
15. Configure the remaining field-level security and page layout settings as desired.
16. Click **Save**.

The Review object now displays the value of the **Position Title** field from the related position record. Next, create a cross-object formula field on the Review object that displays the first and last names of the candidate being reviewed, and we'll use the HYPERLINK function so that users can access the candidate's record by clicking the formula field.

1. Click **Setup > Create > Objects**.
2. Click **Review**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the **Formula** data type, and click **Next**.
5. In the **Field Label** field, enter **Candidate**. Once you move your cursor, the **Field Name** text box should be automatically populated with **Candidate**.

6. Select the Text formula return type and click **Next**.
7. Click the Advanced Formula tab.
8. From the Functions list, double-click HYPERLINK.
9. Delete *url* from the HYPERLINK function you just inserted, but leave your cursor there.
10. Click the **Insert Field** button, select Review >, Job Application >, Candidate >, Record ID, and click **Insert**.
11. Delete *friendly\_name* from the HYPERLINK function, but leave your cursor there.
12. Click the **Insert Field** button, select Review >, Job Application >, Candidate >, First Name, and click **Insert**.
13. Enter a space, click the **Insert Operator** button, and choose Concatenate.
14. Enter another space, then type a blank space enclosed in quotes:

```
" "
```

This appends a blank space after the first name of the candidate.

15. Enter a space, click the **Insert Operator** button, and choose Concatenate once more to add a second ampersand in your formula.
16. Click the **Insert Field** button, select Review >, Job Application >, Candidate >, and Last Name, then click **Insert**.
17. Delete *[ target ]* from the HYPERLINK function. This is an optional parameter that isn't necessary for our formula field.
18. Click **Check Syntax** to check your formula for errors.

Your finished formula should look like this:

```
HYPERNLINK( Job_Application__r.Candidate__r.Id ,
Job_Application__r.Candidate__r.First_Name__c & " " &
Job_Application__r.Candidate__r.Last_Name__c )
```

19. Click **Next**.
20. Configure the remaining field-level security and page layout settings as desired.
21. Click **Save**.

## Discussion

*Cross-object formulas* are formulas that span two or more objects by referencing merge fields from related records. They are available anywhere you can use formulas except for default values and summary reports. Use them in calculations or simply to display fields from related objects on detail pages, list views, related lists, and reports.

Each formula can reference up to five related objects, and can span to an object that is five relationships away. For example, consider the following formula we created in the first set of solution steps:

```
Job_Application__r.Position__r.Name
```

This formula spans two relationships: first it spans to the review's related job application (`Job_Application__r`), then to the job application's related position (`Position__r`). The formula ultimately references the position's title (`Name`) on the Position object. Notice that each part of the formula is separated by a period, and that the relationship names consist of the related object followed by `_r`.

In the second cross-object formula field we created, we used the Concatenate (`&c`) operator to join two separate fields (`First_Name__c` and `Last_Name__c`) and inserted a space between them. We also used the `HYPERNLINK` function, which lets you to create a hyperlink to any URL or record in Salesforce.com. Note that the label of the hyperlink can differ from the URL itself, which is especially useful when working with a cross-object formula field that displays a value that a user will want to click. In this recipe, we used the `HYPERNLINK` function to let users conveniently access the candidate's record by clicking the Candidate's Name field on the Review object.

### See Also

- [Validating Data Based on Fields in Other Records](#) on page 129
- [Averaging Aggregated Data](#) on page 121
- [The Sample Recruiting App](#) on page 3
- “About Formulas” in the Salesforce.com online help
- “Examples of Advanced Formula Fields” in the Salesforce.com online help
- “Formulas: How Do I...” in the Salesforce.com online help
- “Operators and Functions” in the Salesforce.com online help

## Blocking Record Creation with Cross-Object Validation Rules

### Problem

You want to prevent a subset of users from saving a record if certain conditions exist on a related record.

For example, the Recruiting app has the following custom objects:

- Employment Website Information about the cost of posting a position on a particular employment website, such as Monster.com, and the budget the company has allocated for posting on that website.
- Position An open employment opportunity in the company.
- Job Posting A custom junction object between the Employment Website and Position objects that represents a single posting on an employment website.

You want to prevent users from creating a Job Posting record if the record will cause the company to go over its budget for an employment website unless the position was created by the CEO.



**Note:** This recipe has been provided by salesforce.com Training & Certification and is drawn from the expert-led training courses available around the world. Salesforce.com training courses provide an opportunity to get hands-on experience with the Force.com platform and Salesforce.com applications as well as to prepare you to become Salesforce.com Certified. Register for a course at [www.salesforce.com/training](http://www.salesforce.com/training).

## Solution

Create a cross-object validation rule on the Job Posting object that references one roll-up summary field and two currency fields on the Employment Website object.

Create the roll-up summary field.

1. Click **Setup ▶ Create ▶ Objects**.
2. Click **Employment Website**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the **Roll-Up Summary** data type, and click **Next**.
5. In the Field Label field, enter **Current Number of Posts**. Once you move your cursor, the Field Name text box should be automatically populated with **Current\_Number\_of\_Posts**.
6. Click **Next**.
7. In the Summarized Object drop-down list, choose **Job Postings**.
8. Under Select Roll-Up Type, select **COUNT**.
9. Leave All records should be included in the calculation selected, and click **Next**.
10. Configure the remaining field-level security and page layout settings as desired.
11. Click **Save**.

Create the currency field that stores the price per post.

1. Click **Setup ▶ Create ▶ Objects**.
2. Click **Employment Website**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the **Currency** data type, and click **Next**.
5. In the Field Label field, enter **Price Per Post**. Once you move your cursor, the Field Name text box should be automatically populated with **Price\_Per\_Post**.
6. In the Length field, enter 7.
7. In the Decimal Places field, enter 2.
8. Click **Next**.

9. Configure the remaining field-level security and page layout settings as desired.

**10. Click Save.**

Create the currency field that stores the maximum budget.

1. Click **Setup > Create > Objects**.
2. Click **Employment Website**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the **Currency** data type, and click **Next**.
5. In the **Field Label** field, enter **Maximum Budget**. Once you move your cursor, the **Field Name** text box is automatically populated with **Maximum\_Budget**.
6. In the **Length** field, enter **7**.
7. In the **Decimal Places** field, enter **2**.
8. Click **Next**.
9. Configure the remaining field-level security and page layout settings as desired.
- 10. Click Save.**

Create the validation rule.

1. Click **Setup > Create > Objects**.
2. Click **Job Posting**.
3. In the Validation Rules related list, click **New**.
4. In the **Rule Name** field, enter **Max Posts**.
5. Enter the following error condition formula:

```
(  
    Position__r.CreatedBy.UserRole.Name  
    <>  
    "CEO"  
)  
&&  
(  
    Employment_Website__r.Current_Number_of_Posts__c  
    *  
    Employment_Website__r.Price_Per_Post__c  
)  
    >  
(  
    Employment_Website__r.Maximum_Budget__c  
    -  
    Employment_Website__r.Price_Per_Post__c  
)
```

6. In the **Error Message** field, enter **You have exceeded the budget for posting on this employment website.**
7. In the **Error Location** field, select **Top of Page**.
8. Click **Save**.

## Discussion

The first part of the validation rule formula spans to the Positions object to verify that the user who created the position is not the CEO. This gives users the flexibility of going over budget on job postings for positions that the CEO has created.

The second part of the validation rule formula spans to the Employment Website object to retrieve three values that are essential to the calculation. First, the formula references the roll up summary field to count how many Job Posting records have been saved for the associated Employment Website record. Then the formula references the currency field that stores the price per post, and multiplies this value by the job record count to anticipate what the total amount spent on this website will be if the job posting is saved. Finally, the formula references the currency field that stores the maximum budget, and subtracts the price per post from this value. Subtracting the price per post from the maximum budget compensates for the fact that Salesforce.com cannot determine if the record exceeds the budget until after the record is saved.

## See Also

- [Validating Data Based on Fields in Other Records](#) on page 129
- [Averaging Aggregated Data](#) on page 121
- [The Sample Recruiting App](#) on page 3
- “About Validation Rules” in the Salesforce.com online help
- “About Roll-Up Summary Fields” in the Salesforce.com online help
- “Examples of Advanced Formula Fields” in the Salesforce.com online help
- “Considerations for Relationships” in the Salesforce.com online help

# Validating Data Based on Fields in Other Records

## Problem

You want to validate a candidate's ZIP code before saving a candidate record.

## Solution

On the Candidate object in the Recruiting app, create a validation rule that uses the VLOOKUP() function to verify the value of the ZIP Code field against a list of valid ZIP codes stored on a custom object.

1. Create a custom object called ZIP Code with the following settings:

| Field | Value    |
|-------|----------|
| Label | ZIP Code |

| Field                                                              | Value                                                   |
|--------------------------------------------------------------------|---------------------------------------------------------|
| Plural Label                                                       | ZIP Codes                                               |
| Object Name                                                        | ZIP_Code                                                |
| Description                                                        | Represents a ZIP code                                   |
| Context-Sensitive Help Setting                                     | Open the standard Salesforce.com Help & Training window |
| Record Name                                                        | ZIP Code                                                |
| Data Type                                                          | Text                                                    |
| Allow Reports                                                      | No                                                      |
| Allow Activities                                                   | No                                                      |
| Track Field History                                                | No                                                      |
| Deployment Status                                                  | Deployed                                                |
| Add Notes & Attachments<br>related list to default page<br>layout  | No                                                      |
| Launch New Custom Tab Wizard<br>after saving this custom<br>object | Yes                                                     |

2. Add the following custom fields to the ZIP code object:

| Field Label        | Data Type         |
|--------------------|-------------------|
| City               | Text (Length: 20) |
| Latitude           | Number            |
| Longitude          | Number            |
| State              | Text (Length: 20) |
| State Abbreviation | Text (Length: 2)  |

3. Create a validation rule on the Candidate object that uses the following formula:

```
LEN(ZIP_Code__c) > 0 &&
(Country__c = "USA" || Country__c = "US") &&
VLOOKUP(
$ObjectType.ZIP_Code__c.Fields.City__c,
```

```
$ObjectType.ZIP_Code__c.Fields.Name,
LEFT(ZIP_Code__c,5)
<> City__c
)
```

Set the Error Message to The ZIP Code you entered is incorrect.

4. Download the compressed file from [zips.sourceforge.net](http://zips.sourceforge.net). It contains the United States zip codes in Comma-Separated Values (CSV) file format. Extract its contents.
5. Click **Setup > Data Management > Data Loader > Download the Data Loader**.
6. Follow the on-screen instructions to download and install the Data Loader.
7. Use the Data Loader to load the CSV file data into Salesforce.com.
  - a. Launch the Data Loader.
  - b. Click **Insert**.
  - c. Enter your Salesforce.com username and password, and click **Login**.
  - d. After Salesforce.com verifies your login credentials, click **Next**.
  - e. A popup window appears that displays the record count. Click **OK**.
  - f. Select ZIP Code (ZIP\_Code\_c) and click **Next**.
  - g. Click **Create or Edit a Map**.
  - h. Map the Salesforce.com fields to the fields in the CSV file by dragging the Salesforce.com fields from the top table to the bottom table.
  - i. Click **OK**.
  - j. Click **Next**.
  - k. Click **Finish**.
  - l. A popup appears that asks if you want to create new records. Click **Yes**.

## Discussion

The `VLOOKUP()` function returns a value by looking up a related value on a custom object. In this recipe, the validation rule uses the `VLOOKUP()` function to search the `Name` field on all the ZIP code records. It searches until it finds one that matches the value of the `ZIP Code` field on the candidate record that the user is trying to save. After finding the matching ZIP code record, the `VLOOKUP()` function checks the record's `City` field to see if it is not equal to the `City` field on the candidate record. If the search for a matching ZIP code record is unsuccessful, or if the values of the `City` fields on either record do not match, the validation rule prevents the candidate record from being saved, and returns the message The ZIP Code you entered is incorrect.

## See Also

- “About Validation Rules” in the Salesforce.com online help
- [Storing and Displaying Confidential Information](#) on page 118
- “Operators and Functions” in the Salesforce.com online help

- [The Sample Recruiting App](#) on page 3
- “Examples of Advanced Formula Fields” in the Salesforce.com online help

## Using Query String Parameters in a Visualforce Page

### Problem

You want to read and set query string parameters in a Visualforce page, either in a custom controller or in the page itself.

### Solution

The way to read and set query string parameters depends on whether you access them from a custom controller or directly from a Visualforce page.

To read a query string parameter:

- If you're writing a custom controller, use the `ApexPages` global object variable and `currentPage()` and `getParameters()` methods to get query string parameters. For example, to get the value of the name query parameter in the URL: `https://na1.salesforce.com/001/e?name=value`, use the following line in your custom controller:

```
String value = ApexPages.currentPage().getParameters().get('name');
```

- If you're editing a page, use the `$PageContext` global variable in a merge field.

For example, suppose you want to add the Open Activities related list to an account detail page, but instead of showing the account's activities, you want to show the activities of a specified contact. To specify the contact, the following page looks for a query string parameter for the contact's ID under the name `relatedId`:

```
<apex:page standardController="Account">
    <apex:pageBlock title="Hello {!$User.FirstName}!">
        You belong to the {!account.name} account.<br/>
        You're also a nice person.
    </apex:pageBlock>
    <apex:detail subject="{!!account}" relatedList="false"/>
    <apex:relatedList list="OpenActivities"
        subject="{!!$CurrentPage.parameters.relatedId}"/>
</apex:page>
```

For this related list to render in a saved page, valid account and contact IDs must be specified in the URL. For example, if `001D00000HRgU6` is the account ID and `003D000000XDIX` is the contact ID, use the URL

<https://na3.salesforce.com/apex/MyFirstPage?id=001D000000HRgU6&relatedId=003D000000OXDIX>.

To set a query string parameter:

- If you're writing a custom controller, use the `setParameters()` method with `ApexPages.currentPage()` to add a query parameter in a test method. For example:

```
String key = 'name';
String value = 'Caroline';
ApexPages.currentPage().setParameters().put(key, value);
```



**Note:** The `setParameters()` method is only valid inside test methods.

- If you're editing a page, you can either construct a URL manually:

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

Or you can use the `<apex:param>` tag as a child tag to write cleaner code:

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



**Note:** In addition to `<apex:outputLink>`, `<apex:param>` can be a child of other tags such as `<apex:include>` and `<apex:commandLink>`.

## See Also

- [Building a Table of Data in a Visualforce Page](#) on page 69
- [Building a Form in a Visualforce Page](#) on page 71
- [Creating a Wizard with Visualforce Pages](#) on page 72
- [Using AJAX in a Visualforce Page](#) on page 134

## Using AJAX in a Visualforce Page

### Problem

You want to use AJAX in a Visualforce page so that only part of the page needs to be refreshed when a user clicks a button or link.

### Solution

Use the `reRender` attribute on an `<apex:commandLink>` or `<apex:commandButton>` tag to identify the 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, the following page shows a list of contacts. When a user clicks the name of a contact, only the area below the list refreshes, showing the details for the contact:

| Name           | Account Name                        |
|----------------|-------------------------------------|
| Jack Rogers    | Burlington Textiles Corp of America |
| Rose Gonzalez  | Edge Communications                 |
| Pat Stumiller  | Pyramid Construction Inc.           |
| Tim Barr       | Grand Hotels & Resorts Ltd          |
| Stella Pavlova | United Oil & Gas Corp               |
| Barbara Lew    | Express Logistics and Transport     |
| Lauren Boyle   | United Oil & Gas Corp.              |
| John Bond      | Grand Hotels & Resorts Ltd          |
| Andy Young     | Dickenson plc                       |
| Sean Forbes    | Edge Communications                 |

| Contact Detail  |                                                               | <a href="#">Edit</a> | <a href="#">Delete</a>             | <a href="#">Clone</a> | <a href="#">Request Update</a> |
|-----------------|---------------------------------------------------------------|----------------------|------------------------------------|-----------------------|--------------------------------|
| Contact Owner   | Steve Anderson [Change]                                       | Phone                | (312) 596-1000                     |                       |                                |
| Name            | Mr. Tim Barr                                                  | Home Phone           |                                    |                       |                                |
| Account Name    | Grand Hotels & Resorts Ltd                                    | Mobile               | (312) 596-1230                     |                       |                                |
| Title           | SVP, Administration and Finance                               | Other Phone          |                                    |                       |                                |
| Department      | Finance                                                       | Fax                  | (312) 596-1500                     |                       |                                |
| Birthdate       | 3/4/1942                                                      | Email                | barr_tim@grandhotels.com           |                       |                                |
| Reports To      | <a href="#">View Org Chart</a>                                | Assistant            |                                    |                       |                                |
| Lead Source     | External Referral                                             | Asst. Phone          |                                    |                       |                                |
| Mailing Address | 2335 N. Michigan Avenue, Suite 1500<br>Chicago, IL 60601, USA | Other Address        |                                    |                       |                                |
| Languages       | English                                                       | Level                | Secondary                          |                       |                                |
| Created By      | Steve Anderson, 5/28/2008 10:30 AM                            | Last Modified By     | Steve Anderson, 5/28/2008 10:30 AM |                       |                                |
| Description     |                                                               |                      |                                    |                       |                                |
|                 |                                                               | <a href="#">Edit</a> | <a href="#">Delete</a>             | <a href="#">Clone</a> | <a href="#">Request Update</a> |

**Figure 18: Developers Can Use Embedded AJAX to Refresh Part of a Page**

The following markup defines the page from the previous example:

```

<apex:page controller="contactController" showHeader="true"
           tabStyle="Contact">
    <apex:form>
        <apex:dataTable value="{!!contacts}" var="c"
                       cellpadding="4" border="1">
            <apex:column>

```

```

<apex:facet name="header"><b>Name</b></apex:facet>
<apex:commandLink reRender="detail">{!c.name}
    <apex:param name="id" value="{!c.id}" />
</apex:commandLink>
</apex:column>
<apex:column>
    <apex:facet name="header"><b>Account Name</b></apex:facet>
    {!c.account.name}
</apex:column>
</apex:dataTable>
</apex:form>
<apex:outputPanel id="detail">
    <apex:detail subject="{!contact}" title="false"
        relatedList="false"/>
    <apex:relatedList list="ActivityHistories"
        subject="{!contact}" />
</apex:outputPanel>
</apex:page>

```

Notice the following about the markup for this page:

- Setting the `reRender` attribute of the `<apex:commandLink>` tag to 'detail' (the `id` value for the `<apex:outputPanel>` tag) means that only the output panel component is refreshed when a user clicks the name of a contact.
- The `<apex:param>` tag sets the `id` query parameter for each contact name link to the ID of the associated contact record.
- In the `<apex:column>` tags, an `<apex:facet>` tag is used to add the header row. Facets are special child components of some tags that can control the header, footer, or other special areas of the parent component. Even though the columns are in an iteration component (the data table), the facets only display once, in the header for each column.
- In the `<apex:outputPanel>` tag, the details for the currently-selected contact are displayed without the detail section title or complete set of related lists; however, we can add individual related lists with the `<apex:relatedList>` tag.

The following markup defines the Apex controller class for the page. It includes two methods: one to return a list of the ten most recently modified contacts and one to return a single contact record based on the `id` query parameter of the page URL:

```

public class contactController {
    // Return a list of the ten most recently modified contacts
    public List<Contact> getContacts() {
        return [SELECT Id, Name, Account.Name, Phone, Email
                FROM Contact
                ORDER BY LastModifiedDate DESC LIMIT 10];
    }

    // Get the 'id' query parameter from the URL of the page.
    // If it's not specified, return an empty contact.
    // Otherwise, issue a SOQL query to return the contact from the
    // database.
}

```

```
public Contact getContact() {
    Id id = System.currentPageReference().getParameters().get('id');

    return id == null ? new Contact() : [SELECT Id, Name
   FROM Contact
   WHERE Id = :id];
}
```

### See Also

- [Using Query String Parameters in a Visualforce Page](#) on page 132
- [Building a Table of Data in a Visualforce Page](#) on page 69
- [Building a Form in a Visualforce Page](#) on page 71
- [Creating a Wizard with Visualforce Pages](#) on page 72

## Using Properties in Apex

### Problem

You want to create a page that captures input from users, and that input spans multiple sObjects.

### Solution

Use a Visualforce page with a custom Apex controller, and give it properties to represent the input fields from Accounts and Contacts.

1. Create a custom controller with a simple getName method.

- a. Click **Setup > Develop > Apex Classes**.
- b. Click **New**.
- c. In the editor, add the following content:

```
/*
 * This class is the controller for the
 * NewCustomer VisualForce page.
 * It uses properties to hold values entered
 * by the user. These values
 * will used to construct multiple SObjects.
 */
public class Customer {

    // Add properties here

    /* Required method in a VisualForce controller */
    public String getName() {
        return 'Customer';
    }
}
```

```

    }
// Add methods here
// Add queries here
}

```

**d. Click Quick Save.**

- Update the controller by replacing the line // Add properties here with the following Apex properties:

```

public String companyName {get; set;}
public Integer numEmployees {get; set;}
public String streetAddress {get; set;}
public String cityAddress {get; set;}
public String stateAddress {get; set;}
public String postalCodeAddress {get; set;}
public String countryAddress {get; set;}
public String department {get; set;}
public String email {get; set;}
public String phone {get; set;}
public String firstName {get; set;}
public String lastName {get; set;}
public String title {get; set;}

```

**3. Click Quick Save.**

- Update the controller by replacing the line // Add methods here with the following method for saving the property values to a new pair of Account and Contact objects:

```

/*
 * Takes the values entered by the user in the VisualForce
 * page and constructs Account and Contact sObjects.
 */
public void save() {
    Account a = new Account(
        Name = companyName,
        NumberOfEmployees = numEmployees,
        ShippingStreet = streetAddress,
        ShippingCity = cityAddress,
        ShippingState = stateAddress,
        ShippingPostalCode = postalCodeAddress,
        ShippingCountry = countryAddress);

    insert a;

    Contact c = new Contact(
        FirstName = firstName,
        LastName = lastName,
        Account = a,
        Department = department,
        Email = email,
        Phone = phone,
        Title = title,

```

```

        MailingStreet = streetAddress,
        MailingCity = cityAddress,
        MailingState = stateAddress,
        MailingPostalCode = postalCodeAddress,
        MailingCountry = countryAddress);

    insert c;
}

```

5. Click **Quick Save**.
6. Update the controller by replacing the line // Add queries here with queries for displaying Accounts and Contacts related lists:

```

/* Used for the Account list at the end of the
VisualForce page
*/
public List<Account> getAccountList() {
    return [select name, numberofemployees from account];
}

/* Used for the Contact list at the end of the
VisualForce page
*/
public List<Contact> getContactList() {
    return [select name, title, department, email, phone
           from contact];
}

```

7. Click **Save**.
8. Click **SetupDevelop ▶ Pages**.
9. Click **New**.
10. In the name field, enter newCustomerEntry.
11. Optionally enter a label and description.
12. In the editor, enter the following markup:

```

<apex:page controller="Customer">
<apex:form >
    <apex:pageBlock title="New Customer Entry">
        <p>First Name:</p>
        <apex:inputText value="{!!firstName}"/></p>
        <p>Last Name:</p>
        <apex:inputText value="{!!lastName}"/></p>
        <p>Company Name:</p>
        <apex:inputText value="{!!companyName}"/></p>
        <p># Employees:</p>
        <apex:inputText value="{!!numEmployees}"/></p>
        <p>Department:</p>
        <apex:inputText value="{!!department}"/></p>
        <p>Email:</p>
        <apex:inputText value="{!!email}"/></p>
        <p>Phone:</p>
    
```

```

        <apex:inputText value="{!!phone}" /></p>
<p>Title:<br/>
        <apex:inputText value="{!!title}" /></p>
<p>Address</p>
<p>Street:<br/>
        <apex:inputText value="{!!streetAddress}" /></p>
<p>City:<br/>
        <apex:inputText value="{!!cityAddress}" /></p>
<p>State:<br/>
        <apex:inputText value="{!!stateAddress}" /></p>
<p>Zip:<br/>
        <apex:inputText
            value="{!!postalCodeAddress}" /></p>
<p>Country:<br/>
        <apex:inputText value="{!!countryAddress}" /></p>

<p><apex:commandButton action="{!!save}"
            value="Save New Customer" /></p>
</apex:pageBlock>
</apex:form>
<!-- Add related lists here -->
</apex:page>

```

**13. Click Quick Save.**

- 14.** Update the page by replacing <!-- Add related lists here --> with the following markup to displays the related lists from the queries:

```

<apex:pageBlock title="Accounts">
    <apex:pageBlockTable value="{!!accountList}" var="acct">
        <apex:column value="{!!acct.Name}" />
        <apex:column value="{!!acct.NumberOfEmployees}" />
    </apex:pageBlockTable>
</apex:pageBlock>
<apex:pageBlock title="Contacts">
    <apex:pageBlockTable value="{!!contactList}" var="item">
        <apex:column value="{!!item.Name}" />
        <apex:column value="{!!item.Phone}" />
        <apex:column value="{!!item.Title}" />
        <apex:column value="{!!item.Department}" />
        <apex:column value="{!!item.Email}" />
    </apex:pageBlockTable>
</apex:pageBlock>

```

**15. Click Save.**

- 16.** Call the page by using the following URL:

[https://salesforce\\_instance/apex/newCustomerEntry.](https://salesforce_instance/apex/newCustomerEntry)

### Discussion

You want to ask the user to enter information about a new customer. The fields are used to create both a new account and a new contact associated with that account. Using a Visualforce page lets you present whatever user interface you want, using HTML and Visualforce markup; however, each standard controller in Visualforce corresponds to a single sObject type, such as Account or Contact. To work with more than one sObject type, you need to use a custom controller.

When using a Apex custom controller, the easiest way to do to access data exposed by the controller is to use Apex properties. The syntax for Apex properties is similar to C# properties. Java-style bean properties (with getters and setters that you create for each property) also work; however, the property syntax used above is much more readable, and makes it easier to distinguish the controller's properties from its actions.

Queries in a custom controller can be used to present data to the user. In this example, queries are used to create two tables that mimic related lists.

Since the form is simple HTML, you can modify it to your style either using HTML or Visualforce components.

Note that when you add a new customer and click **Save**, the account and contact information is displayed in the related lists on the page.

### See Also

- [Creating a Child Record When a Parent Record is Created](#) on page 159
- [Making Apex Work in any Organization](#) on page 40

## Mass Updating Contacts When an Account Changes

### Problem

You want to update the address of all contacts associated with an account whenever the account's address changes.

### Solution

Write a trigger in Apex that updates associated contacts when an account is updated. For example:

```
trigger updateContactsOnAddressChange on Account  
                                (before update) {  
  
    // The map allows us to keep track of the accounts that have
```

```

// new addresses
Map<Id, Account> acctsWithNewAddresses = new Map<Id, Account>();

// Trigger.new is a list of the Accounts that will be updated
// This loop iterates over the list, and adds any that have new
// addresses to the acctsWithNewAddresses map.
for (Integer i = 0; i < Trigger.new.size(); i++) {
    if ((Trigger.old[i].ShippingCity != Trigger.new[i].
        ShippingCity)
        || (Trigger.old[i].ShippingCountry != Trigger.new[i].
            ShippingCountry)
        || (Trigger.old[i].ShippingPostalCode != Trigger.new[i].
            ShippingPostalCode)
        || (Trigger.old[i].ShippingState != Trigger.new[i].
            ShippingState)
        || (Trigger.old[i].ShippingStreet != Trigger.new[i].
            ShippingStreet)) {
        acctsWithNewAddresses.put(Trigger.old[i].id,
            Trigger.new[i]);
    }
}

List<Contact> updatedContacts = new List<Contact>();

//Here we can see two syntactic features of Apex:
// 1) iterating over an embedded SOQL query
// 2) binding an array directly to a SOQL query with 'in'

for (Contact c : [SELECT id, accountId, MailingCity,
                  MailingCountry, MailingPostalCode,
                  MailingState, MailingStreet
                  FROM contact
                  WHERE accountId
                  in :acctsWithNewAddresses.keySet()]) {
    Account parentAccount = acctsWithNewAddresses.get(c.accountId);

    c.MailingCity = parentAccount.ShippingCity;
    c.MailingCountry = parentAccount.ShippingCountry;
    c.MailingPostalCode = parentAccount.ShippingPostalCode;
    c.MailingState = parentAccount.ShippingState;
    c.MailingStreet = parentAccount.ShippingStreet;

    // Rather than insert the contacts individually, add the
    // contacts to a list and bulk insert it. This makes the
    // trigger run faster and allows us to avoid hitting the
    // governor limit on DML statements
    updatedContacts.add(c);
}
update updatedContacts;
}

```

## See Also

- [Creating a Child Record When a Parent Record is Created](#) on page 159
- [Bulk Processing Records in a Trigger](#) on page 142

- Preventing Duplicate Records from Saving on page 153
- Controlling Recursive Triggers on page 149

## Bulk Processing Records in a Trigger

### Problem

You're new to writing triggers, and when you write one for bulk processing, it often runs into Apex governor limits.

### Solution

For efficient bulk processing, it's critical that triggers execute a constant number of database queries, regardless of how many records are being processed. Instead of looping over individual records in the `Trigger.old` or `Trigger.new` lists, use maps to organize records based on their ID or another identifying field, and use sets to isolate distinct records.

For example, consider the following lead deduplication trigger, which rejects any new or updated lead that has a duplicate email address:

- The trigger first uses a map to store the updated leads with each lead's email address as the key.
- The trigger then uses the set of keys in the map to query the database for any existing lead records with the same email addresses. For every matching lead, the duplicate record is marked with an error condition.

```
trigger leadDuplicatePreventer on Lead
    (before insert, before update) {

    Map<String, Lead> leadMap = new Map<String, Lead>();
    for (Lead lead : System.Trigger.new) {

        // Make sure we don't treat an email address that
        // isn't changing during an update as a duplicate.
        if ((lead.Email != null) &&
            (System.Trigger.isInsert ||
            (lead.Email !=
                System.Trigger.oldMap.get(lead.Id).Email))) {

            // Make sure another new lead isn't also a duplicate
            if (leadMap.containsKey(lead.Email)) {
                lead.Email.addError('Another new lead has the '
                    + 'same email address.');
            } else {
                leadMap.put(lead.Email, lead);
            }
        }
    }
}
```

```
// Using a single database query, find all the leads in
// the database that have the same email address as any
// of the leads being inserted or updated.
for (Lead lead : [SELECT Email FROM Lead
                  WHERE Email IN :leadMap.KeySet()])
{
    Lead newLead = leadMap.get(lead.Email);
    newLead.Email.addError('A lead with this email '
                           + 'address already exists.');
}
```

## See Also

- [Preventing Duplicate Records from Saving](#) on page 153 contains further discussion of the Apex trigger in this recipe.
- [Controlling Recursive Triggers](#) on page 149

# Using Batch Apex to Reassign Account Owners

## Problem

You want to reassign accounts from one owner to another. However, you have 100,000 accounts, which are too many to use a standard Apex trigger or class in a single transaction.

## Solution

Use batch Apex to process all the records at once.

To use batch Apex, you must write a class that implements the `Database.Batchable` interface provided by Salesforce.com.

After you write the class, you should create a Visualforce page for executing the class. You could also create a button that calls a Visualforce controller. It is a best practice to execute a batch job from Visualforce.



**Caution:** You can only have five queued or active batch jobs at one time. Use extreme care if you are planning to invoke a batch job from a trigger. You must be able to guarantee that the trigger will not add more batch jobs than the five that are allowed. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

The following is the entire code example. In the [discussion](#), we include more explanation of the various pieces that must be included in the class.

```
global class AccountOwnerReassignment implements
    Database.Batchable<SObject>, Database.Stateful{

    User fromUser{get; set;}
    User toUser{get; set;}
    Double failedUpdates{get; set;}

    global AccountOwnerReassignment(User fromUser, User toUser) {
        this.fromUser = fromUser;
        this.toUser = toUser;
        failedUpdates = 0;
    }

    global Database.queryLocator
        start(Database.BatchableContext ctx){
            return Database.getQueryLocator([SELECT id, name, ownerId
                FROM Account WHERE ownerId = :fromUser.id]);
        }

    global void execute(Database.BatchableContext ctx, List<Sobject>
        scope){
        List<Account> accs = (List<Account>)scope;

        for(Integer i = 0; i < accs.size(); i++){
            accs[i].ownerId = toUser.id;
        }

        List<Database.SaveResult> dsrs = Database.update(accs, false);

        for(Database.SaveResult dsr : dsrs){
            if(!dsr.isSuccess()){
                failedUpdates++;
            }
        }
    }

    global void finish(Database.BatchableContext ctx){

        AsyncApexJob a = [SELECT id, ApexClassId,
            JobItemsProcessed, TotalJobItems,
            NumberOfErrors, CreatedBy.Email
            FROM AsyncApexJob
            WHERE id = :ctx.getJobId()];

        String emailMessage = 'Your batch job '
            + 'AccountOwnerReassignment '
            + 'has finished. It executed '
            + a.TotalJobItems
            + ' batches. Of which, ' + a.jobitemsprocessed
            + ' processed without any exceptions thrown and '
            + a.NumberOfErrors +
    }
```

```

        ' batches threw unhandled exceptions.'
        + ' Of the batches that executed without error, '
        + failedUpdates
        + ' records were not updated successfully.';

Messaging.SingleEmailMessage mail =
    new Messaging.SingleEmailMessage();
String[] toAddresses = new String[] {a.createdBy.email};
mail.setToAddresses(toAddresses);
mail.setReplyTo('noreply@salesforce.com');
mail.setSenderDisplayName('Batch Job Summary');
mail.setSubject('Batch job completed');
mail.setPlainTextBody(emailMessage);
mail.setHtmlBody(emailMessage);
Messaging.sendEmail(new Messaging.SingleEmailMessage[]
    { mail });
}

public static testmethod void testBatchAccountOwnerReassignment() {
    // Access the standard user profile
    Profile p = [SELECT Id FROM profile
                  WHERE name='Standard User'];

    // Create the two users for the test
    User fromUser = new User(alias = 'newUser1',
                             email='newuser1@testorg.com',
                             emaiencodingkey='UTF-8', lastname='Testing',
                             languagelocalekey='en_US',
                             localesidkey='en_US', profileid = p.Id,
                             timezonesidkey='America/Los_Angeles',
                             username='newuser1@testorg.com');
    User toUser = new User(alias = 'newUser2',
                          email='newuser2@testorg.com',
                          emaiencodingkey='UTF-8', lastname='Testing',
                          languagelocalekey='en_US',
                          localesidkey='en_US', profileid = p.Id,
                          timezonesidkey='America/Los_Angeles',
                          username='newuser2@testorg.com');
    insert fromUser;
    insert toUser;

    // Use the new users to create a new account
    List<Account> accs = new List<Account>();
    for(integer i = 0; i < 200; i++){
        accs.add(new Account(name = 'test',
                             ownerId = fromUser.id));
    }
    insert accs;

    // Actually start the test
    Test.startTest();
    Database.executeBatch(new
        AccountOwnerReassignment(fromUser,
                                  toUser));
    Test.stopTest();
}

```

```

        // Verify the test worked
        accs = [SELECT id, name FROM account
                WHERE ownerId = :toUser.id];
        System.assert(accs.size() == 200);
    }
}

```

## Discussion

The `Database.Batchable` interface provided by Salesforce.com has three methods that must be implemented by your class: `start`, `execute`, and `finish`. This class also includes the method `testBatchAccountOwnerReassignment` used for testing the class. As a best practice you should always test your code, so this example includes testing.

The following steps through each part of the class.

1. Implement the interface `Database.Batchable`.

Each execution of a batch Apex job is considered a discrete transaction. For example, a batch Apex job that contains 1,000 records and is executed without the optional `scope` parameter is considered five transactions of 200 records each. By including `Database.Stateful` in the class declaration, you maintain state across each of these transactions.

```
global class AccountOwnerReassignment implements
    Database.Batchable<SObject>,
    Database.Stateful {
```

2. Declare the variables used in the rest of the class, as well as the class constructor.

```

User fromUser{get; set;}
User toUser{get; set;}
Double failedUpdates{get; set;}

global AccountOwnerReassignment(User fromUser,
                                 User toUser) {
    this.fromUser = fromUser;
    this.toUser = toUser;
    failedUpdates = 0;
}
```

3. The class implementing `Database.Batchable` must implement the `start` method. Use the `start` method to collect the records or objects to be passed to the interface method `execute`. This `start` method populates the `QueryLocator` object with all the accounts owned by the specified owner.

```
global Database.QueryLocator
    start(Database.BatchableContext ctx) {
```

```

        return Database.getQueryLocator([SELECT id,
   name, ownerId
   FROM Account
   WHERE ownerId = :fromUser.id]);
    }
}

```

4. This class must also implement the `execute` method. The `execute` method is called for each batch of records passed to the method. Use this method to do all required processing for each chunk of data. This `execute` method reassigns the owner.

```

global void execute(Database.BatchableContext ctx,
                    List<Sobject> scope){
    List<Account> accs = (List<Account>) scope;

    for(Integer i = 0; i < accs.size(); i++) {
        accs[i].ownerId = toUser.id;
    }

    List<Database.SaveResult> dsrs =
        Database.update(accs, false);
    for(Database.SaveResult dsr : dsrs) {
        if(!dsr.isSuccess()){
            failedUpdates++;
        }
    }
}

```

5. This class must also implement the `finish` method. The `finish` method is called after all batches are processed. This method sends confirmation emails.

```

global void finish(Database.BatchableContext
                   ctx){

    AsyncApexJob a = [SELECT id, ApexClassId,
                      JobItemsProcessed,
                      TotalJobItems,
                      NumberOfErrors,
                      CreatedBy.Email
                      FROM AsyncApexJob
                      WHERE id = :ctx.getJobId()];

    String emailMessage =
        'Your batch job '
        + 'AccountOwnerReassignment'
        + ' has finished. It executed '
        + a.TotalJobItems +
        ' batches. Of which, '
        + a.jobitemsprocessed
        + ' processed without any exceptions'
        + ' thrown and '
        + a.NumberOfErrors +
        ' batches threw unhandled exceptions.'
}

```

```

        + ' Of the batches that executed'
        + 'without error, '
        + failedUpdates
        + ' records were not updated successfully.';

Messaging.SingleEmailMessage mail =
    new Messaging.SingleEmailMessage();
String[] toAddresses = new String[]
    {a.createdBy.email};
mail.setToAddresses(toAddresses);
mail.setReplyTo('noreply@salesforce.com');
mail.setSenderDisplayName('Batch Job Summary');
mail.setSubject('Batch job completed');
mail.setPlainTextBody(emailMessage);
mail.setHtmlBody(emailMessage);
Messaging.sendEmail(new
    Messaging.SingleEmailMessage[] { mail });
}

```

6. Generate the setup data for testing your code. Create the two users that are necessary for the test, as well as the 200 accounts:

```

public static testmethod void
    testBatchAccountOwnerReassignment() {
    // Access the standard user profile
    Profile p = [SELECT Id FROM profile
                 WHERE name='Standard User'];

    // Create the two users for the test
    User fromUser = new User(alias = 'newUser1',
                            email='newuser1@testorg.com',
                            emaiencodingkey='UTF-8',
                            lastname='Testing',
                            languagelocalekey='en_US',
                            localesidkey='en_US', profileid = p.Id,
                            timezonesidkey='America/Los_Angeles',
                            username='newuser1@testorg.com');
    User toUser = new User(alias = 'newUser2',
                            email='newuser2@testorg.com',
                            emaiencodingkey='UTF-8',
                            lastname='Testing',
                            languagelocalekey='en_US',
                            localesidkey='en_US', profileid = p.Id,
                            timezonesidkey='America/Los_Angeles',
                            username='newuser2@testorg.com');
    insert fromUser;
    insert toUser;

    // Use the new users to create a new account
    List<Account> accs = new List<Account>();
    for(integer i = 0; i < 200; i++){
        accs.add(new Account(name = 'test',
                            ownerId = fromUser.id));
    }
    insert accs;
}

```

- The `executeBatch` method starts an asynchronous process. However, your batch job must finish before you can test against the results. Use the Test methods `startTest` and `stopTest` around the `executeBatch` method to ensure the asynchronous process finishes before you test. When the `startTest` method is called, all asynchronous calls get accumulated by the system. When the `stopTest` method is called, all asynchronous jobs are run synchronously.



**Note:** Asynchronous calls, such as `@future` or `executeBatch`, called in a `startTest`, `stopTest` block, do not count against your organization limits for the number of queued jobs.

```
// Actually start the test
Test.startTest();
Database.executeBatch(new
    AccountOwnerReassignment(fromUser, toUser));
Test.stopTest();

// Verify the test worked
accs = [SELECT id, name FROM account
        WHERE ownerId = :toUser.id];
System.assert(accs.size() == 200);
}
```

## See Also

- [Bulk Processing Records in a Trigger](#) on page 142
- [Using Properties in Apex](#) on page 136
- “Batch Apex” in the *Force.com Apex Code Developer’s Guide* available at [www.salesforce.com/us/developer/docs/apexcode/index\\_CSH.htm#apex\\_batch.htm](http://www.salesforce.com/us/developer/docs/apexcode/index_CSH.htm#apex_batch.htm)

# Controlling Recursive Triggers

## Problem

You want to write a trigger that creates a new record as part of its processing logic; however, that record may then cause another trigger to fire, which in turn causes another to fire, and so on. You don’t know how to stop that recursion.

## Solution

Use a static variable in an Apex class to avoid an infinite loop. Static variables are local to the context of a Web request (or test method during a call to `runTests()`), so all triggers that fire as a result of a user’s action have access to it.

For example, consider the following scenario: frequently a Salesforce.com user wants to follow up with a customer the day after logging a call with that customer. Because this is such a common use case, you want to provide your users with a helpful checkbox on a task that allows them to automatically create a follow-up task scheduled for the next day.

You can use a `before insert` trigger on Task to insert the follow-up task, but this, in turn, refires the `before insert` trigger before the follow-up task is inserted. To exit out of this recursion, set a static class boolean variable during the first pass through the trigger to inform the second trigger that it should not insert another follow-up task:



**Note:** For this Apex script to work properly, you first must define a custom checkbox field on Task. In this example, this field is named `Create_Follow_Up_Task__c`.

The following code defines the class with the static class variable:

```
public class FollowUpTaskHelper {  
  
    // Static variables are local to the context of a Web request  
    // (or testMethod during a runTests call)  
    // Therefore, this variable will be initialized as false  
    // at the beginning of each Web request which accesses it.  
  
    private static boolean alreadyCreatedTasks = false;  
  
    public static boolean hasAlreadyCreatedFollowUpTasks() {  
        return alreadyCreatedTasks;  
    }  
  
    // By setting the variable to true, it maintains this  
    // new value throughout the duration of the request  
    // (or testMethod)  
    public static void setAlreadyCreatedFollowUpTasks() {  
        alreadyCreatedTasks = true;  
    }  
  
    public static String getFollowUpSubject(String subject) {  
        return 'Follow Up: ' + subject;  
    }  
}
```

The following code defines the trigger:

```
trigger AutoCreateFollowUpTasks on Task (before insert) {  
  
    // Before cloning and inserting the follow-up tasks,  
    // make sure the current trigger context isn't operating  
    // on a set of cloned follow-up tasks.  
    if (!FollowUpTaskHelper.hasAlreadyCreatedFollowUpTasks()) {
```

```

List<Task> followUpTasks = new List<Task>();

for (Task t : Trigger.new) {
    if (t.Create_Follow_Up_Task__c) {

        // False indicates that the ID should NOT
        // be preserved
        Task followUpTask = t.clone(false);
        System.assertEquals(null, followUpTask.id);

        followUpTask.subject =
FollowUpTaskHelper.getFollowUpSubject(followUpTask.subject);
        if (followUpTask.ActivityDate != null) {
            followUpTask.ActivityDate =
                followUpTask.ActivityDate + 1; //The day after
        }
        followUpTasks.add(followUpTask);
    }
}
FollowUpTaskHelper.setAlreadyCreatedFollowUpTasks();
insert followUpTasks;
}
}

```

The following code defines the test methods:

```

// This class includes the test methods for the
// AutoCreateFollowUpTasks trigger.

public class FollowUpTaskTester {
    private static integer NUMBER_TO_CREATE = 4;
    private static String UNIQUE SUBJECT =
        'Testing follow-up tasks';

    static testMethod void testCreateFollowUpTasks() {
        List<Task> tasksToCreate = new List<Task>();
        for (Integer i = 0; i < NUMBER_TO_CREATE; i++) {
            Task newTask = new Task(subject = UNIQUE_SUBJECT,
                ActivityDate = System.today(),
                Create_Follow_Up_Task__c = true );
            System.assert(newTask.Create_Follow_Up_Task__c);
            tasksToCreate.add(newTask);
        }

        insert tasksToCreate;
        System.assertEquals(NUMBER_TO_CREATE,
            [select count()
             from Task
             where subject = :UNIQUE_SUBJECT
             and ActivityDate = :System.today()]);
    }

    // Make sure there are follow-up tasks created
    System.assertEquals(NUMBER_TO_CREATE,

```

```

        [select count()
         from Task
         where subject =
           :FollowUpTaskHelper.getFollowUpSubject(UNIQUE SUBJECT)
           and ActivityDate = :System.today() +1]];
    }

static testMethod void assertNormalTasksArentFollowedUp() {
    List<Task> tasksToCreate = new List<Task>();
    for (integer i = 0; i < NUMBER_TO_CREATE; i++) {
        Task newTask = new Task(subject=UNIQUE SUBJECT,
                               ActivityDate = System.today(),
                               Create_Follow_Up_Task_c = false);
        tasksToCreate.add(newTask);
    }

    insert tasksToCreate;
    System.assertEquals(NUMBER_TO_CREATE,
                      [select count()
                       from Task
                       where subject=:UNIQUE SUBJECT
                       and ActivityDate =:System.today()]);
}

// There should be no follow-up tasks created
System.assertEquals(0,
                    [select count()
                     from Task
                     where subject=
                       :FollowUpTaskHelper.getFollowUpSubject(UNIQUE SUBJECT)
                     and ActivityDate =:(System.today() +1)]);
}
}

```

## See Also

- Bulk Processing Records in a Trigger on page 142
- Comparing Queries Against Trigger.old and Trigger.new on page 152

# Comparing Queries Against Trigger.old and Trigger.new

## Problem

You're writing a before update or before delete trigger and need to issue a SOQL query to get related data for records in the Trigger.new and Trigger.old lists.

## Solution

Correlate records and query results with the `Trigger.newMap` and `Trigger.oldMap` ID-to-SObject maps.

For example, the following trigger uses `Trigger.oldMap` to create a set of unique IDs (`Trigger.oldMap.keySet()`). The set is then used as part of a query to create a list of job applications associated with the candidates being processed by the trigger. For every job application returned by the query, the related candidate is retrieved from `Trigger.oldMap` and prevented from being deleted.

```
trigger candidateTrigger on Candidate__c (before delete) {
    for (Job_Application__c jobApp : [SELECT Candidate__c
   FROM Job_Application__c
   WHERE Candidate__c
   IN :Trigger.oldMap.keySet()])
    {
        Trigger.oldMap.get(jobApp.Candidate__c).addError(
            'Cannot delete candidate with a job application');
    }
}
```

## Discussion

It's a better practice to use `Trigger.newMap` and `Trigger.oldMap` because you can't assume that directly querying the `Trigger.new` and `Trigger.old` lists will return the same number of records in the same order. Even though these lists are sorted by ID, external operations might change the number of records that are returned and make parallel list processing dangerous.

## See Also

- [Bulk Processing Records in a Trigger](#) on page 142
- [Controlling Recursive Triggers](#) on page 149

# Preventing Duplicate Records from Saving

## Problem

You want to prevent users from saving duplicate records based on the value of one or more fields.

## Solution

If you can determine whether a record is a duplicate based on the value of a single custom field, select the **Unique** and **Required** checkboxes on that field's definition:

- To edit a custom field on a standard object:
  1. Click **Setup** ▶ **Customize**.
  2. Select the link for the desired object, and click **Fields**.
  3. Click **Edit** next to the name of the appropriate field.
- To edit a custom field on a custom object:
  1. Click **Setup** ▶ **Develop** ▶ **Objects**.
  2. Click the name of the object on which the field appears.
  3. Click **Edit** next to the name of the field in the Custom Fields and Relationships related list.

The **Unique** and **Required** checkboxes are only available on custom fields. If you want to check for uniqueness based on the value of a single standard field and your edition can't use Apex, you can also use the following workaround:

1. Create a custom field with the same type and label as the standard field. Select the **Unique** and **Required** checkboxes on the custom field's definition page.
2. Replace the standard field with your new custom field on all page layouts.
3. Use field-level security to make the standard field read-only for all user profiles. This prevents any user from mistakenly modifying the standard field through the API, unless the user has the "Modify All Data" profile permission.
4. Define a workflow rule that automatically updates the value of the standard field with the value of the custom field whenever the custom field changes. This ensures that any application functionality that relies on the value of the standard field continues to work properly. (For example, the **Send An Email** button on the Activity History related list relies on the standard **Email** field for a lead or contact.)



**Note:** Because this is a less-elegant solution than using Apex, creating a trigger on lead is the preferred solution for Unlimited Edition and Developer Edition.

If you need to require uniqueness based on the value of two or more fields, or a single standard field, write an Apex `before insert` and `before update` trigger. For example, the following trigger prevents leads from being saved if they have a matching `Email` field:

- The trigger first uses a map to store the updated leads with each lead's email address as the key.

- The trigger then uses the set of keys in the map to query the database for any existing lead records with the same email addresses. For every matching lead, the duplicate record is marked with an error condition.

```
trigger leadDuplicatePreventer on Lead
    (before insert, before update) {

    Map<String, Lead> leadMap = new Map<String, Lead>();
    for (Lead lead : System.Trigger.new) {

        // Make sure we don't treat an email address that
        // isn't changing during an update as a duplicate.
        if ((lead.Email != null) &&
            (System.Trigger.isInsert ||
            (lead.Email !=
                System.Trigger.oldMap.get(lead.Id).Email))) {

            // Make sure another new lead isn't also a duplicate
            if (leadMap.containsKey(lead.Email)) {
                lead.Email.addError('Another new lead has the '
                    + 'same email address.');
            } else {
                leadMap.put(lead.Email, lead);
            }
        }
    }

    // Using a single database query, find all the leads in
    // the database that have the same email address as any
    // of the leads being inserted or updated.
    for (Lead lead : [SELECT Email FROM Lead
                      WHERE Email IN :leadMap.keySet()])
    {
        Lead newLead = leadMap.get(lead.Email);
        newLead.Email.addError('A lead with this email '
            + 'address already exists.');
    }
}
```

The following class can be used to test the trigger for both single- and bulk-record inserts and updates.

```
public class leadDupePreventerTests{
    static testMethod void testLeadDupPreventer() {

        // First make sure there are no leads already in the system
        // that have the email addresses used for testing
        Set<String> testEmailAddress = new Set<String>();
        testEmailAddress.add('test1@dupitest.com');
        testEmailAddress.add('test2@dupitest.com');
        testEmailAddress.add('test3@dupitest.com');
        testEmailAddress.add('test4@dupitest.com');
        testEmailAddress.add('test5@dupitest.com');
        System.assert([SELECT count() FROM Lead
                      WHERE Email IN :testEmailAddress] == 0);
```

```

// Seed the database with some leads, and make sure they can
// be bulk inserted successfully.
Lead lead1 = new Lead(LastName='Test1', Company='Test1 Inc.',
                      Email='test1@dupitest.com');
Lead lead2 = new Lead(LastName='Test2', Company='Test2 Inc.',
                      Email='test2@dupitest.com');
Lead lead3 = new Lead(LastName='Test3', Company='Test3 Inc.',
                      Email='test3@dupitest.com');
Lead[] leads = new Lead[] {lead1, lead2, lead3};
insert leads;

// Now make sure that some of these leads can be changed and
// then bulk updated successfully. Note that lead1 is not
// being changed, but is still being passed to the update
// call. This should be OK.
lead2.Email = 'test2@dupitest.com';
lead3.Email = 'test3@dupitest.com';
update leads;

// Make sure that single row lead duplication prevention works
// on insert.
Lead dup1 = new Lead(LastName='Test1Dup',
                      Company='Test1Dup Inc.',
                      Email='test1@dupitest.com');
try {
    insert dup1;
    System.assert(false);
} catch (DmlException e) {
    System.assert(e.getNumDml() == 1);
    System.assert(e.getDmlIndex(0) == 0);
    System.assert(e.getDmlFields(0).size() == 1);
    System.assert(e.getDmlFields(0)[0] == 'Email');
    System.assert(e.getDmlMessage(0).indexOf(
        'A lead with this email address already exists.') > -1);
}

// Make sure that single row lead duplication prevention works
// on update.
dup1 = new Lead(Id = lead1.Id, LastName='Test1Dup',
                  Company='Test1Dup Inc.',
                  Email='test2@dupitest.com');
try {
    update dup1;
    System.assert(false);
} catch (DmlException e) {
    System.assert(e.getNumDml() == 1);
    System.assert(e.getDmlIndex(0) == 0);
    System.assert(e.getDmlFields(0).size() == 1);
    System.assert(e.getDmlFields(0)[0] == 'Email');
    System.assert(e.getDmlMessage(0).indexOf(
        'A lead with this email address already exists.') > -1);
}

// Make sure that bulk lead duplication prevention works on
// insert. Note that the first item being inserted is fine,

```

```

// but the second and third items are duplicates. Note also
// that since at least one record insert fails, the entire
// transaction will be rolled back.
dup1 = new Lead(LastName='Test1Dup', Company='Test1Dup Inc.',
                 Email='test4@dupitest.com');
Lead dup2 = new Lead(LastName='Test2Dup',
                 Company='Test2Dup Inc.',
                 Email='test2@dupitest.com');
Lead dup3 = new Lead(LastName='Test3Dup',
                 Company='Test3Dup Inc.',
                 Email='test3@dupitest.com');
Lead[] dups = new Lead[] {dup1, dup2, dup3};
try {
    insert dups;
    System.assert(false);
} catch (DmlException e) {
    System.assert(e.getNumDml() == 2);
    System.assert(e.getDmlIndex(0) == 1);
    System.assert(e.getDmlFields(0).size() == 1);
    System.assert(e.getDmlFields(0)[0] == 'Email');
    System.assert(e.getDmlMessage(0).indexOf(
        'A lead with this email address already exists.') > -1);
    System.assert(e.getDmlIndex(1) == 2);
    System.assert(e.getDmlFields(1).size() == 1);
    System.assert(e.getDmlFields(1)[0] == 'Email');
    System.assert(e.getDmlMessage(1).indexOf(
        'A lead with this email address already exists.') > -1);
}
// Make sure that bulk lead duplication prevention works on
// update. Note that the first item being updated is fine,
// because the email address is new, and the second item is
// also fine, but in this case it's because the email
// address doesn't change. The third case is flagged as an
// error because it is a duplicate of the email address of the
// first lead's value in the database, even though that value
// is changing in this same update call. It would be an
// interesting exercise to rewrite the trigger to allow this
// case. Note also that since at least one record update
// fails, the entire transaction will be rolled back.
dup1 = new Lead(Id=lead1.Id, Email='test4@dupitest.com');
dup2 = new Lead(Id=lead2.Id, Email='test2@dupitest.com');
dup3 = new Lead(Id=lead3.Id, Email='test1@dupitest.com');
dups = new Lead[] {dup1, dup2, dup3};
try {
    update dups;
    System.assert(false);
} catch (DmlException e) {
    System.debug(e.getNumDml());
    System.debug(e.getDmlMessage(0));
    System.assert(e.getNumDml() == 1);
    System.assert(e.getDmlIndex(0) == 2);
    System.assert(e.getDmlFields(0).size() == 1);
    System.assert(e.getDmlFields(0)[0] == 'Email');
    System.assert(e.getDmlMessage(0).indexOf(
        'A lead with this email address already exists.') > -1);
}

```

```

        // Make sure that duplicates in the submission are caught when
        // inserting leads. Note that this test also catches an
        // attempt to insert a lead where there is an existing
        // duplicate.
        dup1 = new Lead(LastName='Test1Dup', Company='Test1Dup Inc.',
                        Email='test4@duptest.com');
        dup2 = new Lead(LastName='Test2Dup', Company='Test2Dup Inc.',
                        Email='test4@duptest.com');
        dup3 = new Lead(LastName='Test3Dup', Company='Test3Dup Inc.',
                        Email='test3@duptest.com');
        dups = new Lead[] {dup1, dup2, dup3};
        try {
            insert dups;
            System.assert(false);
        } catch (DmlException e) {
            System.assert(e.getNumDml() == 2);
            System.assert(e.getDmlIndex(0) == 1);
            System.assert(e.getDmlFields(0).size() == 1);
            System.assert(e.getDmlFields(0)[0] == 'Email');
            System.assert(e.getDmlMessage(0).indexOf(
                'Another new lead has the same email address.') > -1);
            System.assert(e.getDmlIndex(1) == 2);
            System.assert(e.getDmlFields(1).size() == 1);
            System.assert(e.getDmlFields(1)[0] == 'Email');
            System.assert(e.getDmlMessage(1).indexOf(
                'A lead with this email address already exists.') > -1);
        }
    }

    // Make sure that duplicates in the submission are caught when
    // updating leads. Note that this test also catches an attempt
    // to update a lead where there is an existing duplicate.
    dup1 = new Lead(Id=lead1.Id, Email='test4@duptest.com');
    dup2 = new Lead(Id=lead2.Id, Email='test4@duptest.com');
    dup3 = new Lead(Id=lead3.Id, Email='test2@duptest.com');
    dups = new Lead[] {dup1, dup2, dup3};
    try {
        update dups;
        System.assert(false);
    } catch (DmlException e) {
        System.assert(e.getNumDml() == 2);
        System.assert(e.getDmlIndex(0) == 1);
        System.assert(e.getDmlFields(0).size() == 1);
        System.assert(e.getDmlFields(0)[0] == 'Email');
        System.assert(e.getDmlMessage(0).indexOf(
            'Another new lead has the same email address.') > -1);
        System.assert(e.getDmlIndex(1) == 2);
        System.assert(e.getDmlFields(1).size() == 1);
        System.assert(e.getDmlFields(1)[0] == 'Email');
        System.assert(e.getDmlMessage(1).indexOf(
            'A lead with this email address already exists.') > -1);
    }
}

```

## Discussion

The first and most important lesson to learn from this recipe is that you should generally take advantage of point-and-click Force.com functionality if it can solve your problem, rather than writing code. By using the point-and-click tools that are provided, you leverage the power of the platform. Why reinvent the wheel if you can take advantage of a point-and-click feature that performs the same functionality? As a result, we indicate in this recipe that you should first determine whether you can simply use the Unique and Required checkboxes on a single custom field definition to prevent duplicates.

If you do need to check for duplicates based on the value of a single standard field, or more than one field, Apex is the best way to accomplish this. Because Apex runs on the Force.com servers, it's far more efficient than a deduplication algorithm that runs in a Web control. Additionally, Apex can execute every time a record is inserted or updated in the database, regardless of whether the database operation occurs as a result of a user clicking **Save** in the user interface, or as a result of a bulk `upsert` call to the API. Web controls can only be triggered when a record is saved through the user interface.

The included trigger is production-ready because it meets the following criteria:

- The trigger only makes a single database query, regardless of the number of leads being inserted or updated.
- The trigger catches duplicates that are in the list of leads being inserted or updated.
- The trigger handles updates properly. That is, leads that are being updated with email addresses that haven't changed are not flagged as duplicates.
- The trigger has full unit test coverage, including tests for both single- and bulk-record inserts and updates.

## See Also

[Controlling Recursive Triggers](#) on page 149

# Creating a Child Record When a Parent Record is Created

## Problem

You want to automatically create a new child record when you create a parent record. The child record should be populated with default values from the position.

## Solution

Use an Apex trigger to automatically create the child record when a new parent record is created.

For this example, let's automatically create a new interviewer record (child) for the specified hiring manager whenever a new position (parent) is created.

```
trigger AutoCreateInterviewer on Position__c (after insert) {
    List<Interviewer__c> interviewers = new List<Interviewer__c>();

    //For each position processed by the trigger, add a new
    //interviewer record for the specified hiring manager.
    //Note that Trigger.New is a list of all the new positions
    //that are being created.
    for (Position__c newPosition: Trigger.New) {
        if (newPosition.Hiring_Manager__c != null) {
            interviewers.add(new Interviewer__c(
                Name = '1',
                Position__c = newPosition.Id,
                Employee__c = newPosition.Hiring_Manager__c,
                Role__c = 'Managerial'));
        }
    }
    insert interviewers;
}
```

### See Also

- [Bulk Processing Records in a Trigger](#) on page 142
- [Controlling Recursive Triggers](#) on page 149
- [Comparing Queries Against Trigger.old and Trigger.new](#) on page 152

## Using Custom Settings to Display Data

### Problem

You have different default data in an application that you want to display to different users. For example, a system administrator should see one set of data, while a standard user should see a different data set. In addition, you want to be able to quickly access specific sets of data, without worrying about governor limits.

### Solution

Use hierarchy custom settings to specify different default data to different users, and use list custom settings to display different sets of data programmatically. One of the advantages of custom setting data is that it's loaded into cache, and so is quickly available. In addition, accessing the custom setting data sets doesn't require a SOQL or SOSL query, so you won't run into governor limits.

This example has several parts:

- Two Visualforce pages—one for setting up the data, another for displaying it to a user.
- Two Visualforce controllers—written in Apex.
- Two custom setting definitions—one for the list of games, the other for the user preferences.

The following are the complete code examples. See the [discussion](#) on page 166 for detailed walk-throughs of the code and custom settings.

## Visualforce Pages

The administration page:

```
<apex:page Controller="CSAdminController"
            title="Edit my gaming preferences">
    <apex:form>
        <apex:pageblock title="My Game Preferences">

            <apex:pagemessage severity="info" strength="1">
                <!-- Calls getPlatform() in the controller to display
                    the user's GamePref platform based on the
                    hierarchy -->
                Your default platform is: <b>{!Platform}</b>
            </apex:pagemessage>

            <apex:pagemessage severity="info" strength="1">
                <!-- Calls getPlatformValue(User) in the controller to
                    display the user's GamePref platform based on the
                    hierarchy -->
                Your custom preferred platform is: <b>{!PlatformValue}</b>
            </apex:pagemessage>

            <apex:pagemessage severity="info" strength="1">
                <!-- Calls getGenre() in the controller to display the
                    user's GamePref genre based on the hierarchy -->
                Your default genre is: <b>{!Genre}</b>
            </apex:pagemessage>

            <apex:pagemessage severity="info" strength="1">
                <!-- Calls getGenreValue(UserId) in the controller to
                    display the user's GamePref genre based on the
                    hierarchy -->
                Your custom preferred genre is: <b>{!GenreValue}</b>
            </apex:pagemessage>

        </apex:pageblock>

        <apex:pageBlock title="Edit Game Preferences" mode="edit">
            <apex:pageBlockButtons location="top">
                <apex:commandButton action="{!save}" value="Save"/>
            </apex:pageBlockButtons>
            <apex:pageBlockSection
                title="Change my preferences" columns="2">
                <apex:inputField value="{!myPref.platform_c}" />
                <apex:inputField value="{!myPref.genre_c}" />
            </apex:pageBlockSection>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

```

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

```

The user page:

```

<!--Visualforce page to render the custom settings -->
<apex:page controller="CSDemoController" title="What's your game?">
<apex:pageblock >
    <apex:pagemessage severity="info" strength="1">
        <!-- Calls getPlatform() in the controller to display the
            user's GamePref platform based on the hierarchy -->
        Your default platform is: {!Platform}
    </apex:pagemessage>
    <apex:pagemessage severity="info" strength="1">
        <!-- Calls getGenre() in the controller to display the user's
            GamePref genre based on the hierarchy -->
        Your default genre is: {!Genre}
    </apex:pagemessage>

    <apex:pageblocksection title="Sorted by Platform">
        <apex:pageBlockTable value="{!!gamesbyplatform}" var="game"
            id="theTable">
            <apex:column value="{!!game.name}" />
            <apex:column value="{!!game.platform_c}" />
            <apex:column value="{!!game.genre_c}" />
        </apex:pageBlockTable>
    </apex:pageblocksection>

    <apex:pageblocksection title="Sorted by Genre">
        <apex:pageBlockTable value="{!!gamesbygenre}" var="game"
            id="theTable2">
            <apex:column value="{!!game.name}" />
            <apex:column value="{!!game.platform_c}" />
            <apex:column value="{!!game.genre_c}" />
        </apex:pageBlockTable>
    </apex:pageblocksection>
</apex:pageblock>
</apex:page>

```

## Visualforce Controllers

Administration page controller:

```

public class CSAdminController {
    public gameref__c myPref {get;set;}
    public CSAdminController(){
        myPref = gameref__c.getvalues(System.UserInfo.getUserId());
        if(myPref == null)
            myPref = new gameref__c(setupOwnerId =
                System.UserInfo.getUserId());
    }
}

```

```

}

public static String getPlatform() {
    return gamepref__c.getInstance().Platform__c;
}

public static String getGenre(){
    return gamepref__c.getInstance().genre__c;
}

public static String getPlatformValue() {
    if(gamepref__c.getValues(System.UserInfo.getUserId()) == null){
        return 'You do not have a personal gaming preference set';
    }
    else if(gamepref__c.getValues(System.UserInfo.getUserId()).platform__c == null){
        return 'You did not set a personal preferred platform';
    }
    return gamepref__c.getValues(System.UserInfo.getUserId()).Platform__c;
}

public static String getGenreValue(){
    if(gamepref__c.getValues(System.UserInfo.getUserId()) == null){
        return 'You do not have a personal gaming preference set';
    }
    else if(gamepref__c.getValues(System.UserInfo.getUserId()).genre__c == null){
        return 'You did not set a personal preferred genre';
    }
    return gamepref__c.getValues(System.UserInfo.getUserId()).genre__c;
}

public PageReference save() {
    if(myPref.id == null){
        insert myPref;
    }
    else
        update myPref;
    return null;
}
}

```

User page controller:

```

public class CSDemoController {

    public static String getGenre(){
        return gamepref__c.getInstance().genre__c;
    }
}

```

```

public static String getPlatform() {
    return gamepref__c.getInstance().Platform__c;
}

public static List<GameList__c> getGames() {
    return GameList__c.getAll().values();
}

public static List<GameList__c> getGamesByPlatform(){
    List<gamelist__c> AllGames = getGames();
    List<gamelist__c> returnlist = new List<gamelist__c>();
    String myPlatform = getPlatform();
    for(gamelist__c Game : AllGames)
    {
        if(Game.platform__c.contains(myPlatform))
            returnlist.add(Game);
        System.debug(returnlist.size());
    }
    return returnlist;
}

public static List<GameList__c> getGamesByGenre(){
    List<gamelist__c> AllGames = getGames();
    List<gamelist__c> returnlist = new List<gamelist__c>();
    String myGenre = getGenre();
    for(gamelist__c Game : AllGames)
    {
        if(Game.genre__c.contains(myGenre))
            returnlist.add(Game);
    }
    return returnlist;
}

public static TestMethod void test1(){

    Profile SysAdProfileId = [SELECT id FROM Profile
                               WHERE name = 'System Administrator'];

    User newUser = new User
        (username='thecircleoflife@test.com',
        lastName='myLastName', profileId=SysAdProfileId.id,
        email='foobar@a.com', alias='testUser',
        timeZoneSidKey='America/Denver',
        localeSidKey='en_CA', emailEncodingKey='UTF-8',
        languageLocaleKey='en_US');
    insert newUser;

    DemoCSPkg__gamepref__c newUserVal =
        new DemoCSPkg__gamepref__c();
    newUserVal.setupOwnerId = newUser.id;
    newUserVal.platform__c = 'Force.com Platform';
    newUserVal.genre__c = 'Made up genre';
    insert newUserVal;

    DemoCSPkg__GameList__c testGame =
        new DemoCSPkg__GameList__c();
}

```

```
testGame.name = 'Test Game';
testGame.genre__c = 'Made up genre';
testGame.platform__c = 'Platform A';
insert testGame;

DemoCSPkg__GameList__c testGame2 =
    new DemoCSPkg__GameList__c();
testGame2.name = 'Test Game2';
testGame2.genre__c = 'Only the best';
testGame2.platform__c = 'Force.com Platform';
insert testGame2;

System.runAs(newUser)
{
    list<GameList__c> myListGenre = getGamesByGenre();
    list<GameList__c> myListPlatform = getGamesByPlatform();
    for(gamelist__c Game : myListGenre)
    {
        System.Debug(Game.name);
        System.Assert(Game.name.equals('Test Game'));
        System.Assert(Game.genre__c.equals('Made up genre'));

    }
    for(gamelist__c Game : myListPlatform)
    {
        System.Assert(Game.name.equals('Test Game2'));
        System.Assert(Game.platform__c.equals
            ('Force.com Platform'));
    }
}
}
```

## Custom Settings

Hierarchy custom setting:

**Custom Settings** [Help for this Page](#)

Create the fields for your custom settings. The data in these fields is cached with the application.

| Custom Setting Detail |                        | <a href="#">Edit</a> | <a href="#">Delete</a> | <a href="#">Manage</a> |
|-----------------------|------------------------|----------------------|------------------------|------------------------|
| Label                 | gamepref               | Object Name          | gamepref               |                        |
| API Name              | DemoCSPkg__gamepref__c | Setting Type         | Hierarchy              |                        |
| Visibility            | Public                 | Description          |                        |                        |
| Namespace Prefix      | DemoCSPkg              | Created Date         | 6/8/2009 3:46 PM       |                        |
| Last Modified Date    | 6/8/2009 3:46 PM       |                      |                        |                        |

| Custom Fields |             |           |                              | <a href="#">New</a> |
|---------------|-------------|-----------|------------------------------|---------------------|
| Action        | Field Label | Data Type | Modified By                  |                     |
| Edit          | genre       | Text(80)  | Admin User, 6/8/2009 4:09 PM |                     |
| Edit          | platform    | Text(80)  | Admin User, 6/8/2009 4:12 PM |                     |

List custom setting:

**Custom Settings** [Help for this Page](#)

Create the fields for your custom settings. The data in these fields is cached with the application.

| Custom Setting Detail |                        | <a href="#">Edit</a> | <a href="#">Delete</a> | <a href="#">Manage</a> |
|-----------------------|------------------------|----------------------|------------------------|------------------------|
| Label                 | GameList               | Object Name          | GameList               |                        |
| API Name              | DemoCSPkg__GameList__c | Setting Type         | List                   |                        |
| Visibility            | Public                 | Description          |                        |                        |
| Namespace Prefix      | DemoCSPkg              | Created Date         | 6/8/2009 3:46 PM       |                        |
| Last Modified Date    | 6/8/2009 3:46 PM       |                      |                        |                        |

| Custom Fields                      |             |           |                              | <a href="#">New</a> |
|------------------------------------|-------------|-----------|------------------------------|---------------------|
| Action                             | Field Label | Data Type | Modified By                  |                     |
| Edit                               | company     | Text(80)  | Admin User, 6/8/2009 3:57 PM |                     |
| Edit                               | genre       | Text(80)  | Admin User, 6/8/2009 3:55 PM |                     |
| Edit                               | platform    | Text(80)  | Admin User, 6/8/2009 3:56 PM |                     |
| <a href="#">Deleted Fields (1)</a> |             |           |                              |                     |

## Discussion

The first `<apex:pageblock>` on the administrator's Visualforce page calls various Apex methods to return values set by the user. All four are very similar. Let's look more closely at the first two.

Here's the code for the first message on the page:

```
<apex:pagemessage severity="info" strength="1">
<!-- Calls getPlatform() in the controller to display
the user's GamePref platform based on the
hierarchy -->
Your default platform is: <b>{!Platform}</b>
</apex:pagemessage>
```

Here's the corresponding Apex:

```
public static String getPlatform() {
    return gamepref__c.getInstance().Platform__c;
}
```

Custom settings are accessed in Apex just like a custom object. The name of the custom setting being called is gamepref, so in the code it's gamepref\_\_c. The gamepref custom setting is a hierarchy custom setting, which means that the data uses a built-in hierarchical logic that lets you personalize settings for specific profiles or users. The hierarchy logic checks the organization, profile, and user settings for the current user and returns the most specific, or “lowest,” value. In the hierarchy, settings for an organization are overridden by profile settings, which, in turn, are overridden by user settings.

The getInstance method returns the lowest value for the current user for the field Platform defined in the custom setting.

The second method returns the value of the Platform field:

```
<apex:pagemessage severity="info" strength="1">
<!-- Calls getPlatformValue(User) in the controller to
display the user's GamePref platform based on the
hierarchy -->
Your custom preferred platform is: <b>{!PlatformValue}</b>
</apex:pagemessage>
```

Here's the corresponding Apex:

```
public static String getPlatformValue() {
    if(gamepref__c.getValues(System.UserInfo.getUserId()) == null) {
        return 'You do not have a personal gaming preference set';
    }
    else if(gamepref__c.getValues(System.UserInfo.getUserId()).Platform__c == null){
        return 'You did not set a personal preferred platform';
    }
    return gamepref__c.getValues(System.UserInfo.getUserId()).Platform__c;
}
```

If the user has specified a value, it is returned. If the user hasn't specified a value, a message is returned instead.

The second page block of the administrator's page allows the user to set the values for platform and game preference:

```
<apex:pageBlock title="Edit Game Preferences" mode="edit">
    <apex:pageBlockButtons location="top">
        <apex:commandButton action="{!save}" value="Save"/>
    </apex:pageBlockButtons>
    <apex:pageBlockSection title="Change my preferences"
        columns="2">
        <apex:inputField value="{!myPref.platform__c}"/>
        <apex:inputField value="{!myPref.genre__c}"/>
    </apex:pageBlockSection>
</apex:pageBlock>
```

The first page block in the user page that displays the values chosen on the administrator page contains two very similar methods that merely call the controller to retrieve values from the custom setting.

The second and third page blocks in the user page display similar data. Let's look at the first one.

```
<apex:pageblocksection title="Sorted by Platform">
    <apex:pageBlockTable value="{!gamesbyplatform}" var="game"
        id="theTable">
        <apex:column value="{!game.name}"/>
        <apex:column value="{!game.platform__c}"/>
        <apex:column value="{!game.genre__c}"/>
    </apex:pageBlockTable>
</apex:pageblocksection>
```

Here's the corresponding Apex:

```
public static list<GameList__c> getGames() {
    return GameList__c.getAll().values();
}
public static list<GameList__c> getGamesByPlatform() {
    list<GameList__c> AllGames = getGames();
    list<GameList__c> returnlist = new list<GameList__c>();
    String myPlatform = getPlatform();
    for(GameList__c Game : AllGames)
    {
        if(Game.platform__c.contains(myPlatform))
            returnlist.add(Game);
        System.debug(returnlist.size());
    }
    return returnlist;
}
```

The first method returns all the values for the list custom setting gameList as a list. The getAll method returns a map, used with the value method to transform it into a list.

The second method accumulates all the defined values for the user into a list that is then returned to the Visualforce page.

The class CSDDemoController also includes the methods used for testing the class. As a best practice you should always test your code, so this example includes testing.

1. Create the users to test the different roles and data from the hierarchy custom setting:

```
User newUser = new User(username='thecircleoflife@test.com',
    lastName='myLastName', profileId=SysAdProfileId.id,
    email='foobar@a.com', alias='testUser',
    timeZoneSidKey='America/Denver',
    localeSidKey='en_CA', emailEncodingKey='UTF-8',
    languageLocaleKey='en_US');
insert newUser;
```

2. Create new data in both of the custom settings, gamePref and gameList:

```
DemoCSPkg__gamepref__c newUserVal =
    new DemoCSPkg__gamepref__c();
newUserVal.setupOwnerId = newUser.id;
newUserVal.platform__c = 'Force.com Platform';
newUserVal.genre__c = 'Made up genre';
insert newUserVal;

DemoCSPkg__GameList__c testGame = new DemoCSPkg__GameList__c();
testGame.name = 'Test Game';
testGame.genre__c = 'Made up genre';
testGame.platform__c = 'Platform A';
insert testGame;
DemoCSPkg__GameList__c testGame2 = new DemoCSPkg__GameList__c();
testGame2.name = 'Test Game2';
testGame2.genre__c = 'Only the best';
testGame2.platform__c = 'Force.com Platform';
```

3. Run the tests as the specified user:

```
System.runAs(newUser)
{
    list<GameList__c> myListGenre = getGamesByGenre();
    list<GameList__c> myListPlatform = getGamesByPlatform();
    for(gamelist__c Game : myListGenre)
    {
        System.Debug(Game.name);
        System.Assert(Game.name.equals('Test Game'));
        System.Assert(Game.genre__c.equals('Made up genre'));
    }
    for(gamelist__c Game : myListPlatform)
    {
        System.Assert(Game.name.equals('Test Game2'));
    }
}
```

```
System.Assert(Game.platform__c.equals  
('Force.com Platform'));
```

## See Also

- [Using System.runAs in Test Methods](#) on page 170
- [Building a Table of Data in a Visualforce Page](#) on page 69
- “Custom Settings Overview” in the Salesforce.com online help.

# Using System.runAs in Test Methods

## Problem

Generally, all Apex scripts run in system mode. The permissions and record sharing of the current user are not taken into account; however, you need to verify if a specific user has access to a specific object.

## Solution

The system method `runAs` enables you to write test methods that change user contexts to either an existing user or a new user. All of that user's record sharing is then enforced.

In the following example, a new user is created, based on the standard user profile. In addition, a second user is instantiated, based on the system administrator profile, to demonstrate both ways of generating users for tests. Two accounts are created, and then `runAs` verifies that the standard user cannot view the administrator account.

```
@isTest  
private class MyTestClass {  
  
    static testMethod void test1(){  
  
        // Retrieve two profiles, for the standard user and the system  
        // administrator, then populate a map with them.  
  
        Map<String, ID> profiles = new Map<String, ID>();  
        List<Profile> ps = [select id, name from Profile where name =  
            'Standard User' or name = 'System Administrator'];  
  
        for(Profile p : ps){  
            profiles.put(p.name, p.id);  
        }  
  
        // Create the users to be used in this test.  
        // First make a new user.
```

```

User standard = new User(alias = 'standt',
email='standarduser@testorg.com',
emailencodingkey='UTF-8',
lastname='Testing', languagelocalekey='en_US',
localesidkey='en_US',
profileid = profiles.get('Standard User'),
timezonesidkey='America/Los_Angeles',
username='standarduser@testorg.com');

insert standard;

// Then instantiate a user from an existing profile

User admin = [SELECT Id FROM user WHERE profileid =
:profiles.get('System Administrator')];

// Create some test data for testing these two users

List<Account> accnts = new List<Account>();
Account a1 =
    new Account(name='Admin Account', ownerid = admin.id);
Account a2 =
    new Account(name='Standard Account', ownerid = standard.id);

accnts.add(a1);
accnts.add(a2);
insert accnts;

// Confirm that the standard user cannot see the admin account

system.runas(standard){
    accnts.clear();
    accnts = [select id, name from account where id = :a1.id];
    system.debug(accnts.isEmpty() + ' really'+accnts);
    System.assertEquals(accnts.isEmpty(), true);
}

// Confirm that the admin user can see the standard account

system.runas(admin){
    accnts.clear();
    accnts = [select id, name from account where id = :a2.id];
    System.assertEquals(accnts.isEmpty(), false);
}
}

```

## Discussion

Note that this class is defined as `isTest`. Classes defined with the `isTest` annotation do not count against your organization limit of 1 MB for all Apex scripts.

You can only use `runAs` in a test method.

Only the following items use the permissions granted by the user specified with `runAs`:

- dynamic Apex
- methods using `with sharing` or without sharing
- shared records

The original permissions are reset after `runAs` completes.

## See Also

“Understanding Testing in Apex” in the *Force.com Apex Code Developer’s Guide* available at [www.salesforce.com/us/developer/docs/apexcode/index\\_CSH.htm#apex\\_testing\\_intro.htm](http://www.salesforce.com/us/developer/docs/apexcode/index_CSH.htm#apex_testing_intro.htm)

# Integrating Visualforce and Google Charts

## Problem

You want to create a Visualforce page that can render data using Google Charts.

## Solution

Use the `<apex:form>` tag to capture information about the chart. Then, construct the URL that passes the data to the Google Charts API through a custom controller.

## Discussion

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 UnsupportedEncodingTypeException
        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 += '&' + mappedValue;
    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');
    }
}

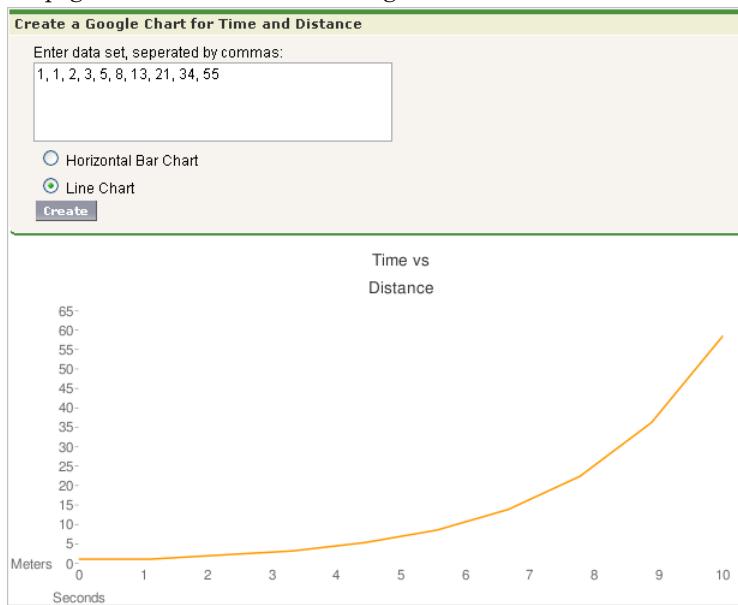
```

```
        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:



## See Also

- [Using Query String Parameters in a Visualforce Page](#) on page 132
- [Using AJAX in a Visualforce Page](#) on page 134

# Using Special Characters in Custom Links

## Problem

You have customers who are using non-English versions of a browser, and URLs and custom links aren't passing special characters properly. The characters either don't show up, or the entire line of code is copied.

## Solution

Encode the URLs with the `encodeURI()` JavaScript function. For example:

```
<script language="JavaScript">
function redirect()
    {parent.frames.location.replace(encodeURI("/003/e?retURL=" +
        "%2F{!Contact.Id}&con4_lkid={!Account.Id}&" +
        "con4 {!Account.Name}&00N30000001KqeH=" +
```

```
"{ !Account.Account_Name_Localized__c}" +  
"&cancelURL=%2F{ !Account.Id}"))}  
redirect();  
</script>
```

# Chapter 5

## Creating Public Websites

---

### In this chapter ...

- [Registering a Custom Domain for Your Force.com Site](#)
- [Using Force.com Site-Specific Merge Fields](#)
- [Customizing the Look and Feel of Your Force.com Site](#)
- [Adding a Feed to Your Force.com Site](#)
- [Creating a Sitemap File](#)
- [Creating a Web-to-Lead Form for Your Force.com Site](#)

Salesforce.com organizations contain valuable information about partners, solutions, products, users, ideas, and other business data. Some of this information would be useful to people outside your organization, but only users with the right access and permissions can view and use it. In the past, to make this data available to the general public, you had to set up a Web server, create custom Web pages (JSP, PHP, or other), and perform API integration between your site and your organization. Additionally, if you wanted to collect information using a Web form, you had to program your pages to perform data validation.

With Force.com sites, you no longer have to do any of those things. Force.com sites enables you to create public websites and applications that are directly integrated with your Salesforce.com organization—without requiring users to log in with a username and password. You can publicly expose any information stored in your organization through a branded URL of your choice. You can also make the site's pages match the look and feel of your company's brand. Because sites are hosted on Force.com servers, there are no data integration issues. And because sites are built on native Visualforce pages, data validation on collected information is performed automatically. You can also enable users to register for or log in to an associated portal seamlessly from your public site.

# Registering a Custom Domain for Your Force.com Site

## Problem

You want to use a branded top-level domain name for your public Force.com site—without exposing the `force.com` domain in the URL.

## Solution

With Sites, you can register a branded domain name for your public site. When users visit your site, they see only the URL that you registered, and not the Force.com domain. For example, if you registered `http://www.mycompany.com` with a domain name registrar, all site traffic is redirected from that domain to your Force.com domain. Users see only `http://www.mycompany.com` in the browser's address bar.

To enable Sites, contact your salesforce.com representative.



**Note:** Custom Web addresses are not supported for sandbox or Developer Edition organizations. SSL is supported, but if org-wide security settings are enabled, and the site-level override is not enabled, branded domains revert to `prefix.secure.force.com` on customer login.

To use a custom Web address for your public site:

1. Register your Force.com domain.
  - a. Click **Setup > Develop > Sites**.
  - b. Enter a unique name for your Force.com domain. 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. Salesforce.com recommends using your company's name or a variation, such as `mycompany`.



**Caution:** You cannot modify your Force.com domain name after you have registered it.

Free Edition domain names are assigned automatically and can't be updated. Create a custom Web address if you want to use a custom domain name.

- c. Click **Check Availability** to confirm that the domain name you entered is unique. If it is not unique, you are prompted to change it.
- d. Read and accept the Sites Terms of Use by selecting the checkbox.

- e. Click **Register My Force.com Domain**. After you accept the Terms of Use and register your Force.com domain, the changes related to site creation are tracked in your organization's Setup Audit Trail and the Site History related list. It may take up to 48 hours for your registration to take effect.

This domain is used for all of your sites, even if you use a branded domain.

2. Register your branded Web address with a domain name registrar, such as GoDaddy.com. For this example, let's say you registered www.acme.com. Once you register the domain, you can also register subdomain names, such as support.acme.com, partners.acme.com, developers.acme.com.
3. Create CNAME records with your registrar to point your top-level domain and your subdomains to your Force.com domain. In this example, create CNAME records to point from:
  - acme.com to acme.force.com
  - support.acme.com to acme.force.com
  - partners.acme.com to acme.force.com
  - acmeideas.com to acme.force.com



**Note:** If you choose to create a branded top-level domain or subdomain through a domain name registrar, the CNAME record that you provide to that registrar must be your Force.com domain name and not the site URL. For example, if you entered mycompany when registering your Force.com domain, the CNAME must be mycompany.force.com, not the full value of the site URL.

4. Create your site and associate your branded domain or subdomain:
  - a. Click **Setup > Develop > Sites**.
  - b. Click **New**.
  - c. On the Site Edit page, define the site and enter your branded domain or subdomain in the **Custom Web Address** field. This field determines which branded domain or subdomain is associated with each site.
  - d. Click **Save**.

## Discussion

By registering a custom Web Address through a domain name registrar and associating it with your site, you can create a completely branded experience for your users. With Sites and Visualforce pages, you can customize the look and feel, the branding, and public security settings for each of your sites.

**See Also**

[Customizing the Look and Feel of Your Force.com Site](#) on page 184

## Using Force.com Site-Specific Merge Fields

**Problem**

You want to create one Visualforce “About Us” page to be used for several public sites, so you can’t hard-code site-specific values like site URL, site email address, and site template. You want the “About Us” page to conditionally display links for registration and login, and connect the “Contact Us” link to each respective site’s designated email address.

**Solution**

Take advantage of the available merge fields when creating the Visualforce pages used for your public sites. By using merge fields, the same Visualforce page can be used for multiple sites.

The following merge fields are available:

| 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.                                                      |
| {!\$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> . |

| Merge Field                     | Description                                                                                                                                                                                                                                |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {!\$Site.AdminEmailAddress}     | Returns the value of the Site Contact field for the current site.                                                                                                                                                                          |
| {!\$Site.Prefix}                | Returns the URL path prefix of the current site. For example, if your site URL is myco.force.com/partners, partners is the path prefix. Returns null 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.                                                                                             |

In the following example, the Visualforce page uses the `{!$Site.Template}`, `{!$Site.AdminEmailAddress}`, `{!$Site.LoginEnabled}`, and `{!$Site.RegistrationEnabled}` merge fields (shown in bold).

```
<apex:page showHeader="false" title="About Us">
<!-- This page uses the template associated with the site.
     No need to hardcode the template name. -->
<apex:composition template="{!$Site.Template}">
    <apex:define name="body">
        <apex:form>
            <apex:outputPanel layout="block">
                <apex:panelGrid columns="1">
                    <apex:outputText value="About Us text goes here"/>
                    <apex:outputText value="..."/>
                    <apex:outputText value="..."/>

<!-- The contact email address is dynamically populated with the
     current site's contact email address. -->
        <apex:panelGroup id="Contact">
            <apex:outputText value="Contact us by email at:"/>
            <apex:outputLink value=
                "mailto:{!$Site.AdminEmailAddress}">
                company@acme.com</apex:outputLink>
```

```
</apex:panelGroup>

<!-- The Login Panel group is rendered if the current site
    is login-enabled. -->
    <apex:panelGroup id="Login" rendered=
        "{!$Site.LoginEnabled}">
        <apex:outputText value="Existing User? "/>
        <apex:outputLink value="{!$Page.SiteLogin}"> Login to
            Access your Account</apex:outputLink>
    </apex:panelGroup> <apex:outputText value="..."/>
        <apex:outputText value="..."/>
        <apex:outputText value="..."/>

<!-- The Register Panel group is rendered if the Customer Portal
    associated with the current site is registration-enabled. -->
    <apex:panelGroup id="Register" rendered=
        "{!$Site.RegistrationEnabled}">
        <apex:outputText value="New User?"/>
        <apex:outputLink value="{!$Page.SiteRegister}">
            Register to get an Account</apex:outputLink>
    </apex:panelGroup>
    </apex:panelGrid>
    </apex:outputPanel>
</apex:form>
</apex:define>
</apex:composition>
</apex:page>
```

## Discussion

Sites takes advantage of the power and flexibility of Visualforce. With the out-of-box merge fields and additional expressions available, you can more easily create feature-rich site pages across multiple sites.

# Customizing the Look and Feel of Your Force.com Site

## Problem

Your Force.com site is up and running, but the pages don't have your company's look and feel. The public sees the standard Salesforce.com beige and blue. You want to use your company's logo and stylesheet.

## Solution

The default sample pages are all branded with the Force.com look and feel, but you can upload your own logo and stylesheet as static resources and reference them from the site template. The site template controls the layout and branding for your entire site.

To brand your site with your logo, colors, and layouts:

1. First, assign the template for your site:
  - a. Click **Setup > Develop > Sites**.
  - b. Click **Edit**.
  - c. Use the lookup field to find and select SiteTemplate for the Site Template field. It may be the default value.
  - d. Click **Save**.
2. Next, upload your stylesheet and company logo as static resources:
  - a. Click **Setup > Develop > Static Resources**.
  - b. For each static resource:
    - i. Click **New**.
    - ii. Enter the name and description, and browse to the file to upload. In this example, the stylesheet is named SiteCSS and the logo is named Logo.
    - iii. Set the **Cache Control** setting to Public. Static resources with private cache control do not show up in sites.
    - iv. Click **Save**.
3. Now modify the SiteHeader to display your company's logo (Logo) and use your company's stylesheet:
  - a. Click **Setup > Develop > Components**.
  - b. Click **Edit** next to the SiteHeader component.
  - c. Replace the following line:
 

```
<apex:image url="{!$Site.Prefix}
{!$Label.site.img_path}/force_logo.gif"
style="align: left;" alt="Salesforce"
width="233" height="55" title="Salesforce"/>
```
  - with this new line:
 

```
<apex:image id="logo" value="{!!$Resource.Logo}"/>
```
  - d. Add the following line after `<apex:image id="logo" value="{!!$Resource.Logo}"/>`:
 

```
<apex:stylesheet value="{!!$Resource.SiteCSS}"/>
```
  - e. Click **Save**.
4. Browse to your site's URL to see your company logo and style.

## Discussion

Enable and customize the Visualforce site template and its components to control the design of your sites. Because the site template can reference other components and static resources, you can change the look and feel of your site with just a few modifications.

## See Also

[Registering a Custom Domain for Your Force.com Site](#) on page 180

# Adding a Feed to Your Force.com Site

## Problem

Your Force.com site is up and running, and you'd like to add a feed that shows your users up to the last 20 accounts added to your Salesforce.com organization.

## Solution

Define a feed and add it to a Visualforce page in your site. Users can then click an icon to subscribe to the feed.

Before you start, make sure that Sites are enabled and that you have set up a site as described in the Salesforce.com online help, including setting the public access settings and enabling feeds for your site.

To create a feed and add it to a Visualforce page in your site:

1. Click **Setup > Develop > Sites** to display the list of sites created for your organization.
2. Click the name of the site where you want to add a feed. Click the `Site Label` link, not the `Site URL` link.
3. In the Syndication Feeds detail area, click **New** and enter the following values:
  - Name: `LatestAccounts`
  - Description: Up to the last 20 accounts added to our organization.
  - Query: `SELECT Name from Account`
  - Mapping ft: "test", fa:"Mysti", et:"Account", ec:Name
  - Max Cache Age Seconds: 600
4. Click **Save**, then click **Back to Site Detail** to return to the sites detail page.

5. Click **Preview** next to **LatestAccounts** to test the feed to ensure it delivers the information you expect. The preview page displays what the feed will display to users, and provides a link to the site where the feed will be displayed.
6. After the feed is created and tested, add a link to the feed in your Visualforce page by adding the following markup:

```
<A HREF="/xml/services/LatestAccounts">Latest Accounts</A>
```

The path assumes your page is located in the base directory of the site. You may have to adjust the path if it is not.

7. Now users who visit the page can click the link **Latest Accounts** and subscribe to the feed.

## Discussion

The feed created here is very simple. You can write much more complex SOQL queries for the feed to tailor the information for your users. For example, the following query would report up the last five accounts created later than yesterday:

```
SELECT Name from Account where CreatedDate < Yesterday LIMIT 5
```

The SOQL query defines what information is collected, but the mapping determines what information is displayed in the feed itself, using elements of the ATOM protocol. For an explanation of the elements, see “Defining Syndication Feeds” in the Salesforce.com online help. This topic also explains some of the limitations placed on SOQL for feeds queries. These limits are in place to ensure good performance.

It's important that you set public access settings for objects properly. Because the feed issues queries as the site guest user, you must assign the correct public access settings to the profile for that guest user, or queries may return either not enough information or information about objects that you don't wish to share with the guest user. Similarly, you must set sharing rules appropriately. Instructions for setting the public access settings and sharing rules are provided in the Salesforce.com online help.

Feeds support the use of bind variables in both the query definition and mapping. At run time, the value of the bind variable is passed in the URL. More information about bind variables is provided in the Salesforce.com online help.

## See Also

- [Visualforce Developer's Guide](#)
- [Customizing the Look and Feel of Your Force.com Site](#) on page 184
- [Registering a Custom Domain for Your Force.com Site](#) on page 180
- [Using Force.com Site-Specific Merge Fields](#) on page 182

## Creating a Sitemap File

### Problem

You want to improve the volume and quality of traffic to your Force.com site.

### Solution

For sites with a large number of dynamic pages that are only available through the use of forms and user entries, you can create a sitemap, which lists the pages on your site that are accessible to Web crawlers and users. The Google, Bing, Yahoo!, and Ask search engines support the sitemaps protocol. For more information on sitemaps and the sitemap protocol, see [www.sitemaps.org](http://www.sitemaps.org).

To create a sitemap, you'll create a Visualforce sitemap page with text or XML content type, and populate the URLs in XML format. Then, to submit the sitemap to Web crawlers, you'll create a `robots.txt` file, adding the path to your sitemap.

First, create the sitemap:

1. Click **Setup** ▶ **Develop** ▶ **Pages**.
2. Click **New**.
3. For the **Label** and **Name**, enter **Sitemap**.
4. Replace the existing Visualforce markup with the following, substituting the example URLs with your site's URLs:

```
<apex:page contentType="text/xml">

<urlset
    xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.sitemaps.org/schemas/sitemap/0.9
        http://www.sitemaps.org/schemas/sitemap/0.9/sitemap.xsd">
    <!-- created with Free Online Sitemap Generator
        www.xml-sitemaps.com -->

    <!-- Replace the following URL with the site URL that you want
        to be mapped and accessible to Web crawlers. -->
    <url>
        <loc>http://mycompany.force.com/mypath</loc>
    </url>
    <!-- Replace the following URLs with your site's actual URLs -->
    <url>
        <loc>http://mycompany.force.com/mypath/pagename</loc>
    </url>
    <url>
        <loc>
```

```

http://mycompany.force.com/mypath/page_detail?id=701300000006CnSAAU
</loc>
</url>
<url>
  <loc>http://mycompany.force.com/mypath/page_index</loc>
</url>
</urlset>
</apex:page>

```



**Note:** Alternatively, you can dynamically create your sitemap page content using a controller, and enable this page for your site.

Then enable the page for your site:

1. Click **Setup > Develop > Sites** and click your site name.
2. In the Site Visualforce Pages list, click **Edit**.
3. Under Available Visualforce Pages, select Sitemap and click **Add**.
4. Click **Save**.

Finally, create and enable the `robots.txt` file:

1. Click **Setup > Develop > Pages**.
2. Click **New**.
3. For the Label and Name, enter `robots`.
4. Replace the existing Visualforce markup with the following, substituting the example URL with your site's path and sitemap name:

```

<apex:page contentType="text/plain">
  User-Agent: *
  Allow: /
  <!-- Replace this path with your site's actual
  path and sitemap name. -->
  Sitemap: http://mycompany.force.com/mypath/Sitemap
</apex:page>

```

5. Click **Setup > Develop > Sites**.
6. Click **Edit** for your site.
7. In the **Site Robots.txt** box, enter `robots`.
8. Click **Save**.



**Note:** Alternatively, you can submit the URL as a sitemap via a [Google Webmaster Tools](#) account.

## Creating a Web-to-Lead Form for Your Force.com Site

### Problem

You want to create a web-to-lead form on your Force.com site to capture prospective customers' personal information. You also want to create a custom contact page that will appear when customers submit their information.

### Solution

Create two Visualforce pages: one to capture a prospect's information and another to display the confirmation. You'll also need to extend the standard controller to redirect public users to a custom contact page when they submit information.

First, create a Visualforce page for the confirmation:

1. Click **Setup > Develop > Pages**.
2. For the Label, enter **Thank You Page** and for the Name, enter **ThankYou**.
3. Replace the existing Visualforce markup with the following:

```
<apex:page title="Job Application"
           showHeader="false" standardStylesheets="true">
    <apex:composition template="{!!$Site.Template}">
        <apex:define name="body">
            <br/>
            <center>
                <apex:outputText value="Your information is saved.
   Thank you for your interest!"/>
            </center>
            <br/>
            <br/>
        </apex:define>
    </apex:composition>
</apex:page>
```

Next, you'll create an extension for the standard controller. By default in Salesforce.com, a standard controller directs the user to the standard detail page after saving the related record. However, standard detail pages can't be made public via sites. To direct users to a custom Visualforce page, you must create an extension for the standard controller. The controller extension creates a new lead record and redirects users to the "Thank You" page. To create the extension:

1. Click **Setup > Develop > Apex Classes**.
2. Click **New**.

- In the editor, add the following content:

```
public class myWeb2LeadExtension {
    private final Lead weblead;
    public myWeb2LeadExtension(ApexPages.StandardController stdController) {
        weblead = (Lead)stdController.getRecord();
    }
    public PageReference saveLead() {
        try {
            insert(weblead);
        }
        catch(System.DMLEException e) {
            ApexPages.addMessages(e);
            return null;
        }
        PageReference p = Page.ThankYou;
        p.setRedirect(true);
        return p;
    }
}
```

Now create a “Contact Us” Visualforce page:

- Click **Setup > Develop > Pages**.
- For the Label, enter Contact Us Page and for the Name, enter ContactUs.
- Replace the existing Visualforce markup with the following:

```
<apex:page standardController="Lead"
           extensions="myWeb2LeadExtension"
           title="Contact Us" showHeader="false"
           standardStylesheets="true">
<apex:composition template="{!$Site.Template}">
<apex:define name="body">
<apex:form>
<apex:messages id="error"
               styleClass="errorMsg"
               layout="table"
               style="margin-top:1em;"/>
<apex:pageBlock title="" mode="edit">
<apex:pageBlockButtons>
<apex:commandButton value="Save"
                     action="{!!saveLead}"/>
</apex:pageBlockButtons>
<apex:pageBlockSection title="Contact Us"
                       collapsible="false"
                       columns="1">
<apex:inputField value="{!!Lead.Salutation}"/>
<apex:inputField value="{!!Lead.Title}"/>
<apex:inputField value="{!!Lead.FirstName}"/>
<apex:inputField value="{!!Lead.LastName}"/>
```

```
<apex:inputField value="{!!Lead.Email}" />
<apex:inputField value="{!!Lead.Phone}" />
<apex:inputField value="{!!Lead.Company}" />
<apex:inputField value="{!!Lead.Street}" />
<apex:inputField value="{!!Lead.City}" />
<apex:inputField value="{!!Lead.State}" />
<apex:inputField value="{!!Lead.PostalCode}" />
<apex:inputField value="{!!Lead.Country}" />
    </apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
</apex:define>
</apex:composition>
</apex:page>
```

Finally, make the pages available on your site:

1. Enable the new Visualforce pages:
  - a. Click **Setup** ▶ **Develop** ▶ **Sites**.
  - b. Click your site label.
  - c. In the Site Details page under Site Visualforce Pages, click **Edit**.
  - d. Select ContactUs and ThankYou and click **Add**.
  - e. Click **Save**.
2. Make sure the fields you exposed on your “Contact Us” page are visible on your site:
  - a. In the Site Details page, click **Public Access Settings**.
  - b. Under Field-Level Security, next to Lead, click **View**.
  - c. Verify that the **Visible** setting is enabled for the Address, Company, Email, Name, and Phone fields. Click **Edit** to change the visibility of any settings.
3. Grant the “Create” permission to your site for the lead object:
  - a. In the Site Details page, click **Public Access Settings**.
  - b. In the Profile page, click **Edit**.
  - c. Under Standard Object Permissions, select the “Create” permission for Leads.
  - d. Click **Save**.
4. Access your “Contact Us” page via your public site and test your new lead form.

## See Also

- [Customizing the Look and Feel of Your Force.com Site](#) on page 184
- [Registering a Custom Domain for Your Force.com Site](#) on page 180

# Chapter 6

## Integrating with Other Applications

---

### In this chapter ...

- [Retrieving Information from Incoming Email Messages](#)
- [Creating Records from Information in Incoming Email Messages](#)
- [Retrieving Email Attachments and Associating Them with Records](#)
- [Creating Email Templates and Automatically Sending Emails](#)
- [Email Recipes—Complete Code Example](#)
- [Updating Salesforce.com Data in the Mobile Application](#)
- [Retrieving a User's Location from a GPS-enabled Phone](#)
- [Enabling Single Sign-On with the Force.com Platform](#)
- [Implementing Single Sign-On for Clients](#)

This chapter contains recipes for integration using email messaging, mobile applications, and delegated authentication. Using Apex classes, functions, and interfaces, you can:

- Automate inbound and outbound message processing with Salesforce.com
- Access and manipulate Salesforce.com data from your mobile device using the Mobile application.
- Create a delegated authentication application to manage single sign-on.

## Retrieving Information from Incoming Email Messages

### Problem

You want to retrieve the following information from a job applicant's incoming email:

- Email address
- First and last name
- Phone number and street address

### Solution

1. Create an Apex class (`ProcessApplicant`) that implements the `InboundEmailHandler` interface. The class defines the `handleInboundEmail` method to work with the created:
  - `InboundEmail`, which contains the incoming email's content, and attachments
  - `InboundEnvelope`, which contains the envelope to and from addresses

For example:

```
global class processApplicant implements
  Messaging.InboundEmailHandler {

  global Messaging.InboundEmailResult handleInboundEmail(
    Messaging.InboundEmail email,
    Messaging.InboundEnvelope envelope) {
  Messaging.InboundEmailResult result =
    new Messaging.InboundEmailresult();

  return result;
}
}
```

2. Parse through the incoming email message and look for the required information.

For example:

```
// Captures the sender's email address
String emailAddress = envelope.fromAddress;

// Retrieves the sender's first and last names
String fName = email.fromname.substring(
  0,email.fromname.indexOf(' '));
String lName = email.fromname.substring(
  email.fromname.indexOf(' '));

// Retrieves content from the email.
// Splits each line by the terminating newline character
// and looks for the position of the phone number and city
String[] emailBody = email.plainTextBody.split('\n', 0);
```

```
String phoneNumber = emailBody[0].substring(6);  
String city = emailBody[1].substring(5);
```

3. Create and activate an email service to process the contents of the `ProcessApplicant` Apex class.

Salesforce.com generates an email address unique to this email service. Emails sent to this inbound email address run the `ProcessApplicant` class:

- a. Click **Setup > Develop > Email Services**.
- b. Click **New Email Service**.
- c. Enter `jobApplication` for the **Email Service Name**.
- d. Enter `ProcessApplicant` for the **Apex Class**.
- e. Click **Save and New Email Address**.
- f. Since the **Email Address** and **Context User** fields have been automatically populated, click **Save**.

## Discussion

- When creating email services, consider security and truncation settings. See “Defining Email Services” in the Salesforce.com online help.
- Email content format varies—ensure that your code can handle different messages by parsing through the entire body of the email.
- For a complete example containing code from all the email recipes, see [Email Recipes—Complete Code Example](#) on page 202.

## See Also

- “Managing Apex Classes” in the Salesforce.com online help
- “Using the `InboundEmail` Object” in the Salesforce.com online help
- “Inbound Email” in the *Force.com Apex Code Developer’s Guide* at [www.salesforce.com/us/developer/docs/apexcode/index\\_CSH.htm#apex\\_classes\\_email\\_inbound.htm](http://www.salesforce.com/us/developer/docs/apexcode/index_CSH.htm#apex_classes_email_inbound.htm)
- “What are Email Services?” in the Salesforce.com online help
- “String Methods” in the *Force.com Apex Code Developer’s Guide* at [www.salesforce.com/us/developer/docs/apexcode/index\\_CSH.htm#apex\\_methods\\_system\\_string.htm](http://www.salesforce.com/us/developer/docs/apexcode/index_CSH.htm#apex_methods_system_string.htm)

# Creating Records from Information in Incoming Email Messages

## Problem

You want to create and associate the following records based on information retrieved from the job applicant's incoming email message:

- Candidate
- Job application

## Solution

Using Apex code, create a new candidate and job application and populate them with the information retrieved from the job applicant's incoming email message. Associate the candidate with the job application. This example assumes that the subject of the job applicant's incoming email exactly matches an existing job name.

```
// Creates new candidate and job application objects
Candidate__c[] newCandidate = new Candidate__c[0];
Job_Application__c[] newJobApplication = new Job_Application__c[0];

// Creates a new candidate from the information
// retrieved from the inbound email
try
{
    newCandidate.add(new Candidate__c(email__c = emailAddress,
        first_name__c = fName,
        last_name__c = lName,
        phone__c = phoneNumber,
        city__c = city));

    insert newCandidate;
}

catch (System.DmlException e)
{
    System.debug('ERROR: Not able to create candidate: ' + e);
}

// Looks for a job name based on the email subject
// (this example assumes the email subject exactly
// matches an existing job name)
Position__c pos;
pos = [select ID from Position__c where name =
    :email.subject limit 1];
ID jobId = pos.ID;

// Associates the candidate with the application
```

```

newJobApplication.add(new Job_Application__c(Position__c = jobId,
candidate__c = newCandidate[0].id));

insert newJobApplication;

```

## Discussion

- This example contains no error handling and assumes the email subject exactly matches the job name. You would need to add error handling and, for example, wildcard matching.
- For a complete example containing code from all the email recipes, see [Email Recipes—Complete Code Example](#) on page 202.

## See Also

- [Retrieving Information from Incoming Email Messages](#) on page 194
- “Using the InboundEmail Object” in the Salesforce.com online help
- “Inbound Email” in the *Force.com Apex Code Developer’s Guide* at [www.salesforce.com/us/developer/docs/apexcode/index\\_CSH.htm#apex\\_classes\\_email\\_inbound.htm](http://www.salesforce.com/us/developer/docs/apexcode/index_CSH.htm#apex_classes_email_inbound.htm)

# Retrieving Email Attachments and Associating Them with Records

## Problem

You want to retrieve an attached resume from the job applicant's incoming email and associate the attachment with the job application.

## Solution

Verify that the incoming email message has an attached file. Create an attachment from the file and associate the attachment with the job application.

This example looks for a binary file attachment (for example, a PDF) that we can associate with a Job Application as a resume:

```

//Searches the email for binary attachments and
// associates them with the job application

if (email.binaryAttachments != null && email.binaryAttachments.size() > 0)
{
    for (integer i = 0 ; i < email.binaryAttachments.size() ; i++)
    {
        Attachment a = new Attachment(ParentId = newJobApplication[0].Id,
Name = email.binaryAttachments[i].filename,

```

```
Body = email.binaryAttachments[i].body);  
insert a;  
}  
}
```

## Discussion

- This example searches for binary attachments only. You would also need to consider other attachment types, such as text file attachments. You can define different actions depending on the type of attached file.
- To view incoming email status and debug any processing errors, view the debug logs created when email services Apex code executes. See “Monitoring Debug Logs” in the Salesforce.com online help.
- For a complete example containing code from all the email recipes, see [Email Recipes—Complete Code Example](#) on page 202.



**Caution:** Attachment validation is an essential part of processing emails. Malicious code can masquerade as a different file type.

## See Also

- “InboundEmail.BinaryAttachment Object” in the *Force.com Apex Code Developer’s Guide* at [www.salesforce.com/us/developer/docs/apexcode/Content/apex\\_classes\\_email\\_inbound\\_binary.htm](http://www.salesforce.com/us/developer/docs/apexcode/Content/apex_classes_email_inbound_binary.htm)
- “Using the InboundEmail Object” in the Salesforce.com online help
- “InboundEmail.TextAttachment Object” in the *Force.com Apex Code Developer’s Guide* at [www.salesforce.com/us/developer/docs/apexcode/Content/apex\\_classes\\_email\\_inbound\\_text.htm](http://www.salesforce.com/us/developer/docs/apexcode/Content/apex_classes_email_inbound_text.htm)

# Creating Email Templates and Automatically Sending Emails

## Problem

You want to send an automatic email response to the job applicant's incoming email message.

## Solution

1. [Create an appropriate email template through the email template wizard.](#) Salesforce.com supports multiple email template types. This examples assumes you are using a Visualforce email template. To create a Visualforce email template, you must have the “Customize Application” permission enabled.

2. Send an automatic email response to the job applicant's incoming email using the `Messaging.sendEmail` static method to process outbound email messages.

First, create a Visualforce email template:

1. Click **Setup > Email > My Templates**. If you have permission to edit public templates, click **Setup > Communication Templates > Email Templates**.
2. Click **New Template**.
3. Choose **Visualforce** and click **Next**.
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 Label**.
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. Optionally, in the **Related To Type** drop-down list, select the object from which the template will retrieve merge field data.
13. Click **Save**.
14. Click **Edit Template**.
15. Enter markup text for your Visualforce email template.
16. 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 cannot exceed 1 MB.

This sample Visualforce email template creates an interview invitation:

```
<messaging:emailTemplate subject="Received your resume"
    recipientType="Contact" relatedToType="Job_Application__c">

<messaging:plainTextEmailBody >
Dear {!relatedTo.Candidate__r.First_Name__c}
    {!relatedTo.Candidate__r.Last_Name__c}

Thank you for your interest in the position
    {!relatedto.Position__r.name}

We would like to invite you for an interview.
```

```
Please respond to the attached invitation.
```

```
Regards,  
Company  
</messaging:plainTextEmailBody>
```

```
<messaging:attachment filename="meeting.ics"  
    renderAs="text/calendar; charset=UTF-8; method=REQUEST">  
BEGIN:VCALENDAR  
METHOD:REQUEST  
BEGIN:VTIMEZONE  
TZID:(GMT-08.00) Pacific Time (US and Canada)  
BEGIN:STANDARD  
DTSTART:16010101T020000  
TZOFFSETFROM:-0700  
TZOFFSETTO:-0800  
RRULE:FREQ=YEARLY;WKST=MO;INTERVAL=1;BYMONTH=11;BYDAY=1SU  
END:STANDARD  
BEGIN:DAYLIGHT  
DTSTART:16010101T020000  
TZOFFSETFROM:-0800  
TZOFFSETTO:-0700  
RRULE:FREQ=YEARLY;WKST=MO;INTERVAL=1;BYMONTH=3;BYDAY=2SU  
END:DAYLIGHT  
END:VTIMEZONE  
BEGIN:VEVENT  
DTSTAMP:20090921T202219Z  
DTSTART;TZID="(GMT-08.00) Pacific Time  
(US and Canada)":20090923T140000  
SUMMARY:Invitation: Interview Schedule @ Wed Sep 23 2pm - 4pm  
ATTENDEE;ROLE=REQ-PARTICIPANT;PARTSTAT=NEEDS-ACTION;RSVP=TRUE;  
CN="{!recipient.name}":MAILTO:{!recipient.email}  
ORGANIZER;CN="John.Smith":MAILTO:recruiter@company.com  
LOCATION:Hawaii  
DTEND;TZID="(GMT-08.00) Pacific Time  
(US and Canada)":20090923T160000  
DESCRIPTION: You are invited to an interview  
    \NInterview Schedule  
    \NWed Sep 23 2pm - 4pm  
    (Timezone: Pacific Time) \NCalendar: John Smith  
    \N\Owner/Creator: recruiter@company.com  
    \NYou will be meeting with these people:  
    CEO Bill Jones,  
    Office Manager Jane Jones  
\N  
SEQUENCE:0  
PRIORITY:5  
STATUS:CONFIRMED  
END:VEVENT  
END:VCALENDAR  
</messaging:attachment>  
</messaging:emailTemplate>
```

Then, send an automatic email response to the job applicant's incoming email. This example uses a Visualforce email template:

```
emailHelper.sendEmail(myContact[0].id, newCandidate[0].id);  
  
    // In a separate class so that it can be used elsewhere  
Global class emailHelper {  
  
    public static void sendEmail(ID recipient, ID candidate) {  
  
        //New instance of a single email message  
        Messaging.SingleEmailMessage mail =  
            new Messaging.SingleEmailMessage();  
  
        // Who you are sending the email to  
        mail.setTargetObjectId(recipient);  
  
        // The email template ID used for the email  
        mail.setTemplateId('00X30000001GLJj');  
  
        mail.setWhatId(candidate);  
        mail.setBccSender(false);  
        mail.setUseSignature(false);  
        mail.setReplyTo('recruiting@acme.com');  
        mail.setSenderDisplayName('HR Recruiting');  
        mail.setSaveAsActivity(false);  
  
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });  
    }  
}
```

## Discussion

- You can use merge fields in Visualforce email templates. Merge fields are placeholders for data that will be replaced with information from your records, user information, or company information.
- To check email delivery status, see “Requesting an Email Log” in the Salesforce.com online help.
- For a complete example containing code from all the email recipes, see [Email Recipes—Complete Code Example](#) on page 202.

## See Also

- “Managing Email Templates” in the Salesforce.com online help
- “Defining Visualforce Pages” in the Salesforce.com online help
- “Creating Visualforce Email Templates” in the Salesforce.com online help
- “Outbound Email” in the *Force.com Apex Developer’s Guide* at [www.salesforce.com/us/developer/docs/apexcode/index\\_CSH.htm#apex\\_classes\\_email\\_outbound.htm](http://www.salesforce.com/us/developer/docs/apexcode/index_CSH.htm#apex_classes_email_outbound.htm)

## Email Recipes—Complete Code Example

The following Apex code is a complete version of the `ProcessApplicant` class, containing code from these recipes:

- [Retrieving Information from Incoming Email Messages](#) on page 194
- [Creating Records from Information in Incoming Email Messages](#) on page 196
- [Retrieving Email Attachments and Associating Them with Records](#) on page 197
- [Creating Email Templates and Automatically Sending Emails](#) on page 198

```
/**  
 * Email services are automated processes that use Apex classes  
 * to process the contents, headers, and attachments of inbound  
 * emails.  
 */  
global class processApplicant implements  
    Messaging.InboundEmailHandler {  
  
    // Creates new candidate and job application objects  
  
    Contact[] myContact = new Contact[0];  
    Candidate__c[] newCandidate = new Candidate__c[0];  
    Job_Application__c[] newJobApplication =  
        new Job_Application__c[0];  
  
    global Messaging.InboundEmailResult handleInboundEmail(  
        Messaging.InboundEmail email,  
        Messaging.InboundEnvelope envelope) {  
        Messaging.InboundEmailResult result =  
            new Messaging.InboundEmailresult();  
  
        // Captures the sender's email address  
        String emailAddress = envelope.fromAddress;  
  
        // Retrieves the sender's first and last names  
        String fName = email.fromname.substring(  
            0,email.fromname.indexOf(' '));  
        String lName = email.fromname.substring(  
            email.fromname.indexOf(' '));  
  
        // Retrieves content from the email.  
        // Splits each line by the terminating newline character  
        // and looks for the position of the phone number and city  
  
        String[] emailBody = email.plainTextBody.split('\n', 0);  
        String phoneNumber = emailBody[0].substring(6);  
        String city = emailBody[1].substring(5);  
  
        // Creates a new candidate from the information  
        // retrieved from the inbound email  
        try {
```

```
{  
    newCandidate.add(new Candidate__c(email__c = emailAddress,  
        first_name__c = fName,  
        last_name__c = lName,  
        phone__c = phoneNumber,  
        city__c = city));  
  
    insert newCandidate;  
}  
  
catch (System.DmlException e)  
{  
    System.debug('ERROR: Not able to create candidate: ' + e);  
}  
  
// Looks for a job name based on the email subject  
// (this example assumes the email subject exactly  
// matches an existing job name)  
Position__c pos;  
pos = [select ID from Position__c where name =  
      :email.subject limit 1];  
ID jobId = pos.ID;  
  
// Associates the candidate with the job application  
newJobApplication.add(new Job_Application__c(  
    Position__c = jobId,candidate__c = newCandidate[0].id));  
  
insert newJobApplication;  
  
// Searches the email for binary attachments and  
// associates them with the job application  
  
if (email.binaryAttachments != null && email.  
    binaryAttachments.size() > 0)  
{  
    for (integer i = 0 ; i < email.binaryAttachments.size() ; i++)  
    {  
        Attachment a = new Attachment(ParentId =  
            newJobApplication[0].Id,  
        Name = email.binaryAttachments[i].filename,  
        Body = email.binaryAttachments[i].body);  
        insert a;  
    }  
}  
  
// Sends the email response  
myContact = createContact.newContact  
    (fName, lName, emailAddress, phoneNumber);  
// Sends an email notification to the applicant  
emailHelper.sendEmail(myContact[0].id, newCandidate[0].id);  
  
return result;  
}
```

```
}

// In a separate class so that it can be used elsewhere
Global class emailHelper {

    public static void sendEmail(ID recipient, ID candidate) {

        //New instance of a single email message
        Messaging.SingleEmailMessage mail =
            new Messaging.SingleEmailMessage();

        // Who you are sending the email to
        mail.setTargetObjectId(recipient);

        // The email template ID used for the email
        mail.setTemplateId('00X30000001GLJj');

        mail.setWhatId(candidate);
        mail.setBccSender(false);
        mail.setUseSignature(false);
        mail.setReplyTo('recruiting@acme.com');
        mail.setSenderDisplayName('HR Recruiting');
        mail.setSaveAsActivity(false);

        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });

    }
}
```

## Updating Salesforce.com Data in the Mobile Application

### Problem

You want to modify related lists of Salesforce.com records in the mobile application, even when the records are not available locally on the mobile phone. Optionally, you'll use a Bluetooth peripheral device to update the data in the records.

### Solution

For this recipe, let's assume that your organization sells servers. Your field service representatives want to update related list information on records from their mobile devices while at customer sites. For example, if they replace a broken component in a server, they want to be able to enter the new component's serial number and update the server record. The server is then associated with the new component instead of the replaced one.

To accomplish this, we'll create three custom objects and some custom fields. We'll use an Apex trigger to update the server record when the field service representatives replace a component.

Before starting this recipe, make sure you meet the following requirements:

- **Mobile Licenses:** All existing Developer Edition organizations have mobile licenses. Visit [Developer Force](#) and click **Getting Started** to sign up for a new Developer Edition organization.
- **Mobile Devices:** Verify that your mobile device will run Salesforce Mobile. See Salesforce Mobile Supported Devices in the [Salesforce.com online help](#).

If your mobile device is not supported, you can download and install the [BlackBerry simulator package](#) so you can run the mobile application on a simulator.

To use Apex to update data in Salesforce Mobile:

First, create a custom object named Server that has a lookup relationship to Account. This object allows your organization to keep track of the servers housed at each customer's site.

1. Click **Setup** ▶ **Create** ▶ **Objects** ▶ **New Custom Object**.
2. Enter the following information:
  - Label: Server
  - Plural Label: Servers
3. Accept the remaining defaults and click **Save**.
4. In the Custom Fields & Relationships related list, click **New** and define a field with the following attributes:
  - Data Type: Lookup Relationship
  - Related To: Account
  - Field Label: Account
  - Field Name: Account
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts
5. In the Custom Fields & Relationships related list, click **New** and define a second field with the following attributes:
  - Data Type: Text Area
  - Field Label: Description
  - Field Name: Description
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts

Next, create a custom object named Component that has a lookup relationship to Server. This object represents the various components that can be installed inside a server.

1. Click **Setup > Create > Objects > New Custom Object**.
2. Enter the following information:
  - Label: Component
  - Plural Label: Components
3. Accept the remaining defaults and click **Save**.
4. In the Custom Fields & Relationships related list, click **New** and define a field with the following attributes:
  - Data Type: Master-Detail Relationship
  - Related To: Server
  - Field Label: Server
  - Field Name: Server
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts
5. In the Custom Fields & Relationships related list, click **New** and define a second field with the following attributes:
  - Data Type: Checkbox
  - Field Label: Operational
  - Field Name: Operational
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts
6. In the Custom Fields & Relationships related list, click **New** and define a third field with the following attributes:
  - Data Type: Picklist
  - Field Label: Type
  - Values:Motherboard, RAM, CPU, Network Card, Power Supply
  - Field Name: Type
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts
7. By default, one of the standard fields for the Component object is Component Name; however, the field service representatives reference the components by serial number. We'll change the standard field name to Component Serial Number. In the Standard Fields related list, click **Edit** next to **Component Name**.
8. In the Record Name field, type Component Serial Number.
9. Click **Save**.

Next, create a custom object named Replacement. This object allows the field service representatives to enter information about the new component.

1. Click **Setup > Create > Objects > New Custom Object**.
2. Enter the following information:
  - Label: Replacement
  - Plural Label: Replacements
3. Accept the remaining defaults and click **Save**.
4. In the Custom Fields & Relationships related list, click **New** and define a field with the following attributes:
  - Data Type: Date
  - Field Label: Date
  - Field Name: Date
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts
5. In the Custom Fields & Relationships related list, click **New** and define a second field with the following attributes:
  - Data Type: Lookup Relationship
  - Related To: Component
  - Field Label: New Component
  - Field Name: New Component
  - Field-Level Security: Hidden from all profiles
  - Add Related List: Hidden from all page layouts
6. In the Custom Fields & Relationships related list, click **New** and define a third field with the following attributes:
  - Data Type: Lookup Relationship
  - Related To: Component
  - Field Label: Replaced Component
  - Field Name: Replaced Component
  - Field-Level Security: Hidden from all profiles
  - Add Related List: Hidden from all page layouts

In the next two steps, we'll create text versions of the New Component and Replaced Component lookup fields. This is because Salesforce Mobile handles lookup fields differently than the Salesforce.com website. Users can type in lookup fields on the website, but not in the mobile application. We want the field representatives to be able to quickly enter the serial number of the new component, so we'll create text fields to replace the lookup fields.

To further complicate matters, mobile configurations deliver only a subset of a user's data to the mobile application. There is a possibility that the record the user is trying to look up is not stored locally on the mobile device. By allowing a user to enter text, the mobile application can send the information to Salesforce.com when the record is saved, and then Salesforce.com validates the user's text entry against the component records with the Apex trigger.

Additionally, if we want to pair the mobile device with a Bluetooth barcode scanner so that the field representatives can simply scan the serial numbers of the new and replaced components, it is imperative to set up the fields as text fields. For information about using a barcode reader, see the [Discussion](#) at the end of this recipe.

7. In the Custom Fields & Relationships related list, click **New** and define a fourth field with the following attributes:
  - Data Type: Text
  - Field Label: New Component Serial Number
  - Field Name: New Component Serial Number
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts
8. In the Custom Fields & Relationships related list, click **New** and define a fifth field with the following attributes:
  - Data Type: Text
  - Field Label: Replaced Component Serial Number
  - Field Name: Replaced Component Serial Number
  - Field-Level Security: Visible for all profiles
  - Add Related List: On all page layouts

Next, create an Apex trigger on the Replacement object so that a server's components are automatically updated when a field representative creates a replacement record.

1. Click **Setup** > **Create** > **Objects** > **Replacement**.
2. In the Triggers related list, click **New**.
3. In the Body text box, enter the following Apex trigger code:

```
trigger replacementPart on Replacement__c (before insert) {
    for(Replacement__c r : Trigger.new) {
        //first, find the old component
        //(the one we are replacing)
        Component__c oldCompInfo = [select id, type__c, Server__c
            from component__c where Name=
            :r.Replaced_Component_Serial_Number__c];
```

```

//find the new component by name
Component__c newCompInfo = [select id, type__c from
    component__c where
        Name=:r.New_Component_Serial_Number__c];
if(oldCompInfo == null)
    r.Replaced_Component_Serial_Number__c.addError(
        'The serial number you entered does not match '+
        'any existing components.');
if(newCompInfo == null)
    r.New_Component_Serial_Number__c.addError(
        'The serial number you entered does not match '+
        'any existing components.');
//set the Replacement part's fields
r.Replaced_Component__c = oldCompInfo.Id;
r.New_Component__c = newCompInfo.Id;
if(newCompInfo.type__c != oldCompInfo.type__c)
    r.New_Component__c.addError(
        'The new component type must be ' +
        'the same as the replaced component type.');
//update the old component's Server__c to null
Component__c oldComp = new Component__c(
    Id= oldCompInfo.Id, Server__c=null);
//update the new component's Server__c
//to the old component's Server__c
component__c newComp = new Component__c(
    Id=r.New_Component__c, Server__c=
        oldCompInfo.Server__c);
//update both components
List<Component__c> comps = new List<Component__c>();
comps.add(oldComp);
comps.add(newComp);
update(comps);
r.Date__c = System.Today();
}
}
}

```

#### 4. Click Save.

Next, create a mobile configuration and mobilize the new objects.

1. Click **Setup > Mobile Configurations > New Mobile Configuration**.
2. In the Name field, type **Field Service**.
3. Select the **Active** checkbox.
4. Select the **Mobilize Recent Items** checkbox.
5. Select your name in the Available Members list box and click **Add**.
6. Click **Save**.
7. In the Data Sets related list, click **Edit**.
8. Click **Add....**
9. Select **Server** and click **OK**.
10. Click the **Data Sets** node in the tree, click **Add....** Select **Component**, and click **OK**.

- 11.** Click the **Data Sets** node in the tree, and click **Add....** Select **Replacement**, and click **OK**.

If this was a real mobile configuration for field representatives, you would probably also mobilize other objects, like Account and Case. For the recipe, we just need to mobilize the custom objects we created.

- 12. Click Done.**

Finally, test the Apex trigger in the mobile application:

- 1.** On the Salesforce.com website, create a server record and three related component records so you have some data to work with in the mobile application.
  - Server: IBM BladeCenter S
  - Component Serial Number: 100000; Type: CPU
  - Component Serial Number: 100001; Type: Motherboard
  - Component Serial Number: 100002; Type: Power Supply
- 2.** Create one more component record that is not associated with any server record. Enter 100003 as the Component Serial Number, and set the Type to CPU.
- 3.** Install the mobile application on your device. For installation instructions, see “Installing the Mobile Application” in the Salesforce.com online help. If you’re using a BlackBerry simulator, the mobile application is already installed on the simulator device; you just need to activate your Salesforce.com Developer Edition account after launching Salesforce Mobile for the first time.
- 4.** In the mobile application, select the Replacement tab.
- 5.** Open the menu and select **New**.
- 6.** In the Replaced Component Serial Number field, type 100000.
- 7.** In the New Component Serial Number field, type 100003.



**Tip:** Pairing the mobile device with a Bluetooth barcode scanner allows the field service representative to enter the serial numbers in the replacement record fields by scanning the barcodes.

- 8.** Open the menu and select **Save**.

The mobile application sends the record to Salesforce.com and the new component is associated with the server record.

## Discussion

To make this solution even more effective, implement the Apex trigger and give the field representatives barcode scanners so that they can quickly scan the serial numbers when replacing components. Many barcode scanners can operate wirelessly using a Bluetooth connection. Just

pair the barcode scanner with a Bluetooth-enabled mobile device, and you'll be able to use the scanner for entering data in the mobile application.

### See Also

- For information about extending your Visualforce pages to mobile users, see [Retrieving a User's Location from a GPS-enabled Phone](#) on page 211.
- For information about deploying the mobile solution to Salesforce.com users, see the *Salesforce Mobile Implementation Guide* in the Salesforce.com online help.

## Retrieving a User's Location from a GPS-enabled Phone

### Problem

You want to capture the location where your mobile users enter Salesforce.com data by retrieving the GPS coordinates from a BlackBerry smartphone or iPhone when users save a record.

### Solution

Write a Visualforce page that sales representatives can use when logging sales visits. The Visualforce page contains JavaScript that captures the device's longitude and latitude when the record is saved. After writing the Visualforce page, we'll create a tab for the page and add the tab to the mobile application.

Before starting this recipe, make sure you complete the following prerequisites:

- Mobile Licenses:** All existing Developer Edition organizations have mobile licenses. Visit [Developer Force](#) and click **Getting Started** to sign up for a new Developer Edition organization.
- Mobile Devices:** Verify that your BlackBerry smartphone or iPhone can use Salesforce Mobile. See "Salesforce Mobile Supported Devices" in the Salesforce.com online help.
- GPS Receiver:** Verify that your BlackBerry smartphone has an internal GPS receiver or an external Bluetooth GPS receiver, and that the receiver is enabled. To find out if your BlackBerry smartphone's GPS receiver is on, select **Options > Advanced > GPS** or **Options > Advanced > Location Based Services**. If your device does not meet these requirements, download and install the [BlackBerry simulator package](#).

GPS is unavailable in first-generation iPhones. The earliest iPhone model with GPS support is the iPhone 3G.

First, create a custom object named Sales Visit. This object allows sales representatives to enter information in Salesforce.com after making a sales visit.

- Click **Setup > Create > Objects > New Custom Object**.

2. Enter the following information:
  - Label: Sales Visit
  - Plural Label: Sales Visits
  - Description: Include a brief description so other developers know what this object does.
3. Accept the remaining defaults and click **Save**.
4. In the Custom Fields & Relationships related list, click **New** and define a field with the following attributes:
  - Data Type: Text Area
  - Field Label: Description
  - Field Name: Description
  - Field-Level Security: Visible for all profiles
  - Add Related List: Select for all page layouts
5. In the Custom Fields & Relationships related list, click **New** and define a second field with the following attributes:
  - Data Type: Number
  - Field Label: Longitude
  - Length: 3
  - Decimal Places: 10
  - Field Name: Longitude
  - Field-Level Security: Visible for all profiles
  - Add Related List: Select for all page layouts
6. In the Custom Fields & Relationships related list, click **New** and define a third field with the following attributes:
  - Data Type: Number
  - Field Label: Latitude
  - Length: 3
  - Decimal Places: 10
  - Field Name: Latitude
  - Field-Level Security: Visible for all profiles
  - Add Related List: Select for all page layouts

Create an Apex class that mimics the Sales Visit object but changes the save behavior:

1. Click **Setup** ▶ **Develop** ▶ **Apex Classes**, then click **New**.

**2.** Enter the following code:

```
public class visitController {
    public Sales_Visit__c visit {get;set;}
    public visitController() {
        visit = new Sales_Visit__c();
    }
    public PageReference save() {
        insert visit;
        visit = new Sales_Visit__c();
        return null;
    }
}
```

Normally after saving a record, the record's detail page displays, along with the tabs and sidebar. The Apex class suppresses all these screen elements—which would overload the mobile device's small screen—by displaying a blank Sales Visit record instead.

**3. Click Save.**

Create a Visualforce page that sends the user's GPS coordinates when the sales visit record is saved.

1. Click **Setup > Develop > Pages > New.**
2. In the Label field, enter **Sales Visit Form**.
3. In the Name field, enter **Sales\_Visit\_Form**.
4. Replace the generic markup with the following Visualforce component:

```
<apex:page controller="visitController" showHeader="false"
    sidebar="false" setup="true" standardStylesheets="false"
    id="ServiceForm">
<head>
    <meta name="viewport" content="width = device-width"/>
    <script type="text/javascript"
        src="/mobileclient/api/mobileforce.js"></script>
    <script>
        function updateLocation(lat,lon) {
            document.getElementById(
                {!$Component.form.block.longitude}).value=lon;
            document.getElementById(
                {!$Component.form.block.latitude}).value=lat;
        }
        function getLocation() {
            mobileforce.device.getLocation(updateLocation);
            //work around required for BB
            if (window.blackberry)
                setInterval("getLocation()", 10000);
            return false;
        }
    </script>

```

```

</script>
</head>

<apex:form id="form">
    <apex:pageBlock id="block">
        Sales Visit Name: <br />
        <apex:inputField value="{!visit.name}" /><br />
        Sales Visit Description: <br />
        <apex:inputField value="{!visit.Description__c}" /><br />
        Longitude: <br />
        <apex:inputField value="{!visit.Longitude__c}"
            id="longitude" /><br />
        Latitude: <br />
        <apex:inputField value="{!visit.Latitude__c}"
            id="latitude" /><br />
        <button type="button" value="GPS"
            onclick="getLocation();"> Get location </button>
        <apex:commandButton action="{!save}" value="Save!" />
    </apex:pageBlock>
</apex:form>

```

## 5. Click **Save**.

Create a tab for the Visualforce page.

1. Click **Setup** ▶ **Create** ▶ **Tabs** ▶ **New**.
2. Select Sales\_Visit\_Form from the Visualforce Page drop-down.
3. In the Tab Label field, enter Sales Visit.
4. Select a tab style.
5. Select the **Mobile Ready** checkbox.
6. Click **Next**.
7. Click **Next** again.
8. Click **Save**.

Create a mobile configuration and mobilize the new Visualforce page.

1. Click **Setup** ▶ **Mobile Configurations** ▶ **New Mobile Configuration**.
2. In the Name field, type Sales Representative.
3. Select the Active checkbox.
4. Select the Mobilize Recent Items checkbox.
5. Select your name in the Available Members list box, and then click **Add**.
6. Click **Save**.
7. In the Data Sets related list, click **Edit**.
8. Click **Add....**
9. Select **Account** and click **OK**.
10. Click the **Data Sets** node in the tree and click **Add....** Select **Opportunity** and click **OK**.
11. Click the **Data Sets** node in the tree and click **Add....** Select **Lead** and click **OK**.

**12. Click Done.**

13. In the Mobile Tabs related list, click **Customize Tabs**.

14. Select the Account, Opportunity, Lead, and Sales Visit tabs and click **Add**.

If the Sales Visit tab isn't in the Available Tabs list, you might not have selected the **Mobile Ready** checkbox when creating the Visualforce tab.

**15. Click Save.**

Test the Visualforce page in the mobile application.

1. Install the mobile application on your device. For installation instructions, see "Installing the Mobile Application" in the Salesforce.com online help. If you're using a BlackBerry simulator, the mobile application is already installed on the simulator device; just activate your Salesforce.com Developer Edition account after launching Salesforce Mobile for the first time.

If you already completed [Updating Salesforce.com Data in the Mobile Application](#) on page 204, you don't need to reinstall the application. Simply open the mobile application and refresh the data on your device. To synchronize the data:

- On a BlackBerry smartphone, open the main menu, and select **System Info**. Open the menu and select **Refresh All Data**.
  - On an iPhone, tap **More**, then tap **App Info**. Tap **Sync Now** on the toolbar, then tap **Refresh All Data**.
2. In the mobile application, select the Sales Visit tab.
  3. Open the menu, and select **New**.
  4. In the **Description** field, enter a description of the sales visit.
  5. Open the menu and select **Save**.

The record is saved and sent to Salesforce.com along with the user's GPS location.

The application permissions on the BlackBerry smartphone control whether or not the browser is allowed to send the GPS coordinates. If the browser is not permitted to send the information, the Visualforce page doesn't display. To ensure that all mobile Salesforce.com users are able to properly view the page, the BlackBerry administrator should globally enable the application permissions on the BlackBerry Enterprise Server. To enable the permission on your BlackBerry smartphone:

- a. Select **Options > Advanced Options > Applications**.
- b. Highlight **Browser**, open the main menu, and select **Edit Default Permissions**.
- c. Change the value of **Location (GPS)** to **Allow**.

## Discussion

The Visualforce markup in this recipe uses commands in a JavaScript library provided by Salesforce.com to obtain the device's GPS coordinates. The JavaScript library contains commands that trigger actions in Salesforce Mobile, which helps provide a seamless user experience between Visualforce Mobile pages and the native client application.

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 actions in the 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.com recommends that version 4.6 or later of the BlackBerry operating system is installed on the device.

For more information, see “Using the JavaScript Library” in the [Visualforce Developer’s Guide](#).

## Enabling Single Sign-On with the Force.com Platform

### Problem

You want to validate usernames and passwords for Salesforce.com against your corporate user database or another client application rather than having separate user passwords managed by Salesforce.com.

### Solution

Salesforce.com offers two ways to use single sign-on:

#### Delegated Authentication

Use delegated authentication if you have mobile users in your organization, or if you want to enable single-sign on for partner portals or Customer Portals.

You must request that this feature be enabled by salesforce.com. This recipe explains delegated authentication in more detail.

## Federated Authentication using SAML

Federated authentication uses SAML, an industry standard for secure integrations. Investing in SAML with Salesforce.com can be leveraged with other products or services. If you use SAML, you don't have to expose an internal server to the Internet: the secure integration is done using the browser. In addition, Salesforce.com never handles any passwords used by your organization.

For more information, see “Configuring SAML Settings for Single Sign-On” in the Salesforce.com online help.

When delegated authentication is enabled, salesforce.com does not validate a user's password. Instead, salesforce.com makes a Web services call to your organization to establish authentication credentials for the user.

When implementing delegated authentication, select one of the following security modes:

### Simple passwords

Users directly provide passwords that the delegated authentication server evaluates. In this mode, users can also use the Salesforce.com login page and all of the Salesforce.com clients without modification or extra infrastructure.

### Tokens

Instead of passwords, security tokens are provided, and the delegated authentication server evaluates these. This mode hides all passwords from Salesforce.com. Instead of a user signing onto a Salesforce.com login page, the user signs onto a private login page on your company's web server that may be behind your corporate firewall. In this mode, the user is authenticated on that private login page. When the user has been successfully authenticated, the user is redirected to the Salesforce.com login page and logged in using a single-use token. In this mode, users can't login directly to the Salesforce.com login page (users can't generate tokens), and they can't directly login using a client or mobile app. This recipe uses this mode.

### Mixed

Your organization creates a delegated authentication server that can evaluate either tokens or passwords. This enables organizations to mobile and client apps, while still providing their users with a private login page.



**Note:** Contact salesforce.com to enable delegated authentication for your organization if it is not already enabled.

To enable delegated authentication for your organization, build your delegated authentication Web service, and then configure your Salesforce.com organization to enable the Web service.

First, build your delegated authentication Web service:

1. In Salesforce.com, click **Setup > Develop > API > Download Delegated Authentication WSDL** to download the Web Services Description Language (WSDL) file, AuthenticationService.wsdl.

The WSDL describes the delegated authentication service and can be used to automatically generate a server-side stub to which you can add your specific implementation. For example, in the WSDL2Java tool from Apache Axis, you can use the --server-side switch. In the wsdl.exe tool from .NET, you can use the /server switch.

A simple C# implementation of delegated authentication is shown below. The sample uses Microsoft .NET v1.1 and works with IIS6 on a Windows 2003 server. See [wiki.apexdevnet.com/index.php/How\\_to\\_Implement\\_Single\\_Sign-On\\_with\\_Force.com](http://wiki.apexdevnet.com/index.php/How_to_Implement_Single_Sign-On_with_Force.com) for a more complex sample that demonstrates a solution using an authentication token.

```
using System;
using System.DirectoryServices;

namespace samples.sforce.com
{
    /// <summary>
    /// This is a very basic implementation of an
    /// authentication service for illustration purposes.
    /// This sample should only be used
    /// with a HTTPS Delegated Gateway URL.
    /// It simply connects to your Active Directory
    /// server using the credentials that are passed in.
    /// If there is a bad username/password combination,
    /// it throws an exception and returns false;
    /// otherwise the credentials are ok
    /// and it returns true.
    /// Note that DirectoryEntry might not connect to
    /// Active Directory until we do something
    /// that actually requires it.
    /// That's why we read a property from the
    /// created DirectoryEntry object.
    /// </summary>
    public class SimpleAdAuth : System.Web.Services.WebService
    {
        [System.Web.Services.WebMethodAttribute()]
        [System.Web.Services.Protocols.SoapDocumentMethodAttribute(
            "",
            RequestNamespace = "urn:authentication.soap.sforce.com",
            ResponseElementName = "AuthenticateResult",
            ResponseNamespace = "urn:authentication.soap.sforce.com",
            Use = System.Web.Services.Description.SoapBindingUse.Literal,
            ParameterStyle =
            System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        [return:
            System.Xml.Serialization.XmlElementAttribute("Authenticated")]
    }
}
```

```
public bool Authenticate (
    string username, string password, string sourceIp,
    [System.Xml.Serialization.XmlAnyElementAttribute()]
    System.Xml.XmlElement[] Any)
{
    if(username.IndexOf("@") == -1)
        return false;

    // Attempt to bind to an Active Directory entry.
    // This will authenticate the username and password.
    // TODO: you'll need to change this to match
    // your Active Directory name
    const string root = "LDAP://DC=sample,DC=org";
    try
    {
        DirectoryEntry de
            = new DirectoryEntry(root, username, password);
        // retrieve a property
        string tempName = de.Name;
        return true;
    }
    catch(Exception)
    {
        return false;
    }
}
```

As part of the delegated authentication process, a salesforce.com server makes a SOAP 1.1 request to authenticate the user who is passing in the credentials. An example of this type of request is shown below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Authenticate xmlns="urn:authentication.soap.sforce.com">
      <username>sampleuser@sample.org</username>
      <password>myPassword99</password>
      <sourceIp>1.2.3.4</sourceIp>
    </Authenticate>
  </soapenv:Body>
</soapenv:Envelope>
```

Your delegated authentication service needs to accept this request, process it, and return a `true` or `false` response. A sample response is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
  <AuthenticateResponse
  xmlns="urn:authentication.soap.sforce.com">
```

```
<Authenticated>true</Authenticated>
</AuthenticateResponse>
</soapenv:Body>
</soapenv:Envelope>
```

2. Add a link to your corporate intranet or other internally-accessible site that takes the authenticated user's credentials and passes them using an HTTPS POST to the Salesforce.com login page. For security reasons, you should make your service available by SSL only. This ensures that a password, if it is included, is not sent unencrypted. You must use an SSL certificate from a trusted provider, such as Verisign or Thawte.

Because Salesforce.com does not use the password field other than to pass it back to you, you do not need to send a password in this field. Instead, you could pass another authentication token, such as a Kerberos Ticket, so that your actual corporate passwords are not passed to or from Salesforce.com.

You can configure the Salesforce.com-delegated authentication authority to allow only tokens, or to accept either tokens or passwords. If the authority only accepts tokens, a Salesforce.com user cannot log in to Salesforce.com directly, because they cannot create a valid token; however, many companies choose to allow both tokens and passwords. In this environment, a user can still log in to Salesforce.com through the login page.

When the salesforce.com server passes these credentials back to you in the **Authenticate** message, verify them, and the user will gain access to the application.

Next, in Salesforce.com, specify your organization's delegated authentication gateway URL by clicking **Setup** ▶ **Security Controls** ▶ **Single Sign-On Settings** ▶ **Edit**. Enter the URL in the **Delegated Gateway URL** text box. For security reasons, Salesforce.com restricts the outbound ports you may specify to one of the following:

- 80: This port only accepts HTTP connections.
- 443: This port only accepts HTTPS connections.
- 7000-10000 (inclusive): These ports accept HTTP or HTTPS connections.



**Note:** For security reasons, you should make your service available by SSL only.

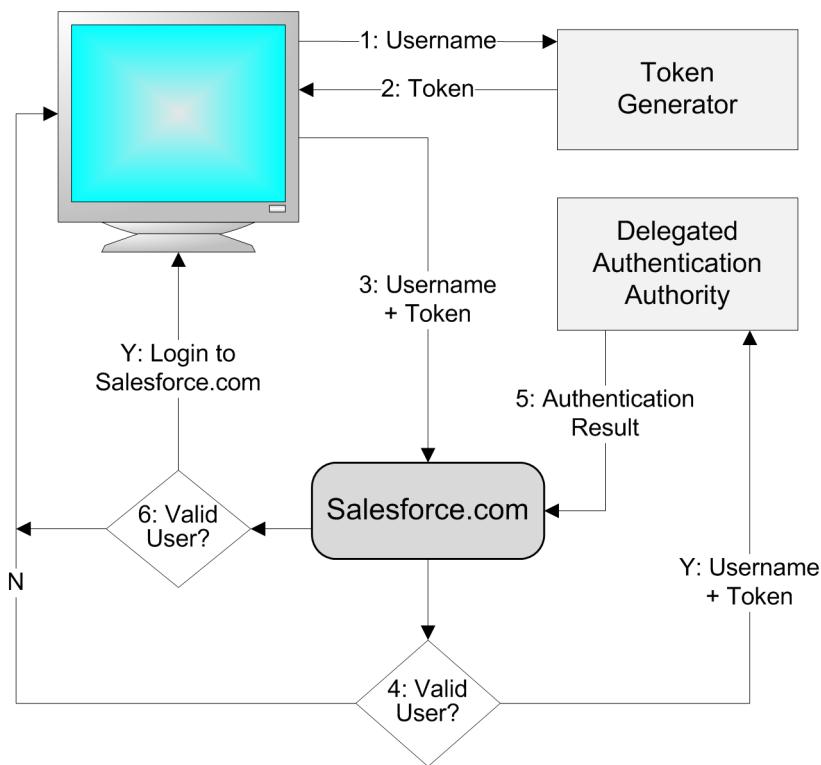
Finally, modify your user profiles to enable the **Is Single Sign-On Enabled** user permission. In Salesforce.com, click **Setup** ▶ **Manage Users** ▶ **Profiles** to add or edit profiles.

### Discussion

The actual implementation of delegated authentication is transparent to users, but involves a number of steps behind the scenes. When you configure Salesforce.com for delegated

authentication, you are allowing a delegated authority to control authentication. When a user first logs onto their network environment, they are initially authenticated by this authority. When the user attempts to log on to subsequent protected applications, instead of passing a username and password to the application, the user requests a token from a token generator. (On Windows, this token request can use the NTLM protocol.) The received token is passed to the application, which verifies that the token properly identifies the user, and then allows the user access to the application.

Salesforce.com can use this method, since the password field is simply used to exchange information with the client, rather than specifying a particular data type. This flexibility means that Salesforce.com can accept a token, which is then used with the delegated authentication authority to verify the user. If the verification succeeds, the user is logged on to Salesforce.com. If the verification fails, the user receives an error. The process flow for Salesforce.com delegated authentication is shown in the figure below.



**Figure 19: Delegated Authentication Process Flow**

## See Also

- *How to Implement Single Sign-On with the Force.com at*  
[wiki.apexdevnet.com/index.php/How\\_to\\_Implment\\_Single\\_Sign-On\\_with\\_Force.com](http://wiki.apexdevnet.com/index.php/How_to_Implment_Single_Sign-On_with_Force.com)

- “Configuring SAML Settings for Single Sign-On” in the Salesforce.com online help
- *The Single Sign-On Implementation Guide* in the Salesforce.com online help

## Implementing Single Sign-On for Clients

### Problem

You want to use single sign-on with a desktop client, such as Outlook Edition, Lotus Notes Edition, or Office Edition, so that your users only have to log into an environment once—instead of having to login more than once to access different services and resources in an environment.



**Note:** This recipe assumes that you are familiar with enabling single sign-on.

### Solution

There are three main approaches for using single sign-on to authenticate Force.com clients:

- Use the network password, such as an LDAP password for authentication using the clients. End users would enter their Salesforce.com usernames and LDAP password into the login dialog box, and delegated authentication would be performed.



**Note:** Force.com never logs the LDAP password, and clears it out of memory as soon as it has passed on in the SOAP message to the single sign-on service.

- Use a client application registry setting that can designate where Force.com directs the login request. By making this URL an internal URL, a customer can provide a proxy for the username and password, verify it locally, then pass a one-time use token (such as a SAML token) to Force.com for verification. This is then passed back to the customer for validation.
- Use a customer-built proxy that requires NT Lan Manager (NTLM) authentication. Once NTLM has passed, the proxy can send the Salesforce.com username and a one-time use token to Force.com, which gets passed back to the customer for validation. This approach has the benefit of not having to configure a username and password for all clients that are deployed. Only the registry setting needs to be changed.

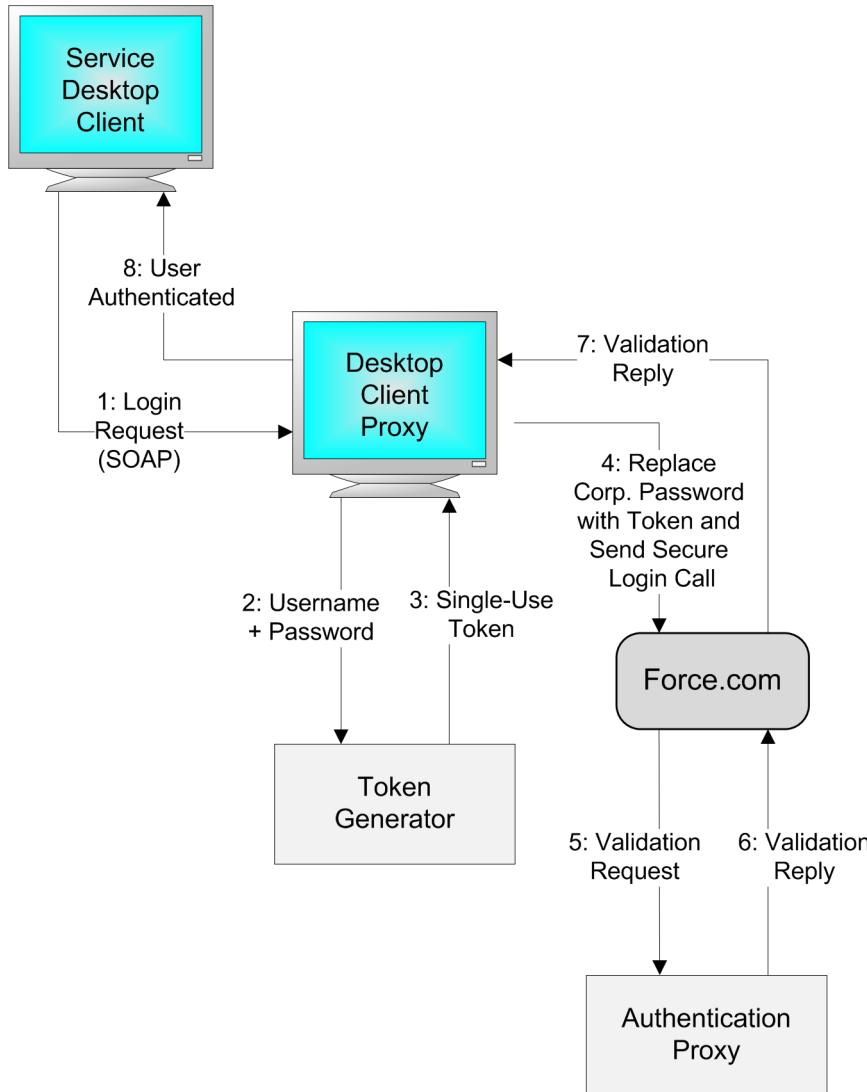


**Note:** You do not need to implement a proxy service to make client applications work if your single sign-on listener supports tokens and passwords. Users can configure their passwords in the client application. In this case, the network password temporarily passes through the Force.com servers. This password is not logged anywhere, and is cleared out of memory as soon as the outbound SOAP message has been sent.

## Discussion

The single sign-on login process for Force.com desktop clients involves the following components:

- SOAP message packages
- Local Microsoft® Windows registry key  
HKEY\_LOCAL\_MACHINE\SOFTWARE\salesforce.com\OfficeToolkit\ServerUrl
- Desktop client proxy (specified in the registry key)
- Token generator
- Single use tokens
- Token authentication proxy

**Figure 20: Single Sign-On Login Process from a Client**

1. The desktop client sends a login request to the desktop client proxy as a SOAP message package.
2. The desktop client proxy extracts the username and password and sends them to the token generator.
3. The token generator validates the credentials and replies to the desktop client proxy with a single-use token.

4. The desktop client proxy modifies the SOAP message package by replacing the corporate password in the login request with the token and sends a secure login call to Force.com.
5. Force.com sends a request to the authentication proxy to validate the token.
6. The authentication proxy replies to Force.com.
7. Force.com replies to the desktop client proxy.
8. The desktop client proxy passes the response back to the desktop client, authenticating the user.



**Note:** This process occurs only if the registry key has a value.

When the desktop client proxy is not exposed to the public Internet, only users inside the network or with VPN access have the ability to use authenticate to Force.com using the client applications.

## Sample Messages

The following is an example of a HTML/SOAP login message. As summarized in [the first step](#) above, login messages such as the following sample are sent to the desktop client proxy specified in the `ServerUrl` registry key.

```

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sfdc="urn:enterprise.soap.sforce.com"
  xmlns:sf="urn:sobject.enterprise.soap.sforce.com">
  <soapenv:Header>
    <sfdc:CallOptions>
      <sfdc:client>Outlook/33101</sfdc:client>
      <sfdc:remoteApplication>outlook</sfdc:remoteApplication>
    </sfdc:CallOptions>
    <sfdc:QueryOptions>
      <sfdc:batchSize>100</sfdc:batchSize>
    </sfdc:QueryOptions>
  </soapenv:Header>
  <soapenv:Body>
    <sfdc:login>
      <sfdc:username xsi:type="xsd:string">b.lake@salesforce.com
      </sfdc:username>
      <sfdc:password xsi:type="xsd:string">proxy1234
      </sfdc:password>
    </sfdc:login>
  </soapenv:Body>
</soapenv:Envelope>
```

The following is an example of a SOAP response message. As summarized in [the last step](#) above, responses from Force.com such as the following sample are passed by the desktop client proxy to the desktop client.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="urn:enterprise.soap.sforce.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <loginResponse>
            <result>
                <metadataServerUrl>https://na1-api.salesforce.com/services/Soap
                    /m/10.0/00D30000000Kc5B</metadataServerUrl>
                <passwordExpired>false</passwordExpired>
                <serverUrl>https://na1-api.salesforce.com/services/Soap
                    /c/10.0/00D30000000Kc5B</serverUrl>
                <sessionId>00D30000000Kc5B!ARKAQPsowLYLaw8G_Y1AgnUG9mZ3Z1mSi
                    NCKis1Q691trluugmUqstbFdLUCddq2PFTFZRxsXRXx0aBS82XJME6x
                    MP_4Xut6</sessionId>
                <userId>00530000001Yx3rAAC</userId>
                <userInfo><accessibilityMode>false</accessibilityMode>
                    <currencySymbol>$</currencySymbol>
                    <organizationId>00D30000000Kc5BERS</organizationId>
                    <organizationMultiCurrency>false
                        </organizationMultiCurrency>
                    <organizationName>San Francisco Coffee Supply
                        </organizationName>
                    <profileId>00e30000000w2Z8AAI</profileId>
                    <roleId xsi:nil="true"/>
                    <userDefaultCurrencyIsoCode xsi:nil="true"/>
                    <userEmail>bjmark@salesforce.com</userEmail>
                    <userFullName>Blake J Mark</userFullName>
                    <userId>00530000001Yx3rAAC</userId>
                    <userLanguage>en_US</userLanguage>
                    <userLocale>en_US</userLocale>
                    <userName>blake@desktopclientsdemo.com</userName>
                    <userTimeZone>America/Los_Angeles</userTimeZone>
                    <userType>Standard</userType>
                    <userUiSkin>Theme2</userUiSkin>
                </userInfo>
            </result>
        </loginResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

## See Also

- [Enabling Single Sign-On with the Force.com Platform](#) on page 216
- *How to Implement Single Sign-On with Force.com* at [wiki.apexdevnet.com/index.php/How\\_to\\_Implment\\_Single\\_Sign-On\\_with\\_Force.com](http://wiki.apexdevnet.com/index.php/How_to_Implment_Single_Sign-On_with_Force.com)
- “Configuring SAML Settings for Single Sign-On” in the Salesforce.com online help
- *The Single Sign-On Implementation Guide* in the Salesforce.com online help

# Chapter 7

## Integrating Applications with the API and Apex

---

### In this chapter ...

- Setting Up Your Salesforce.com Web Services API Applications
- Implementing the query()/queryMore() Pattern
- Batching Records for API Calls
- Using a Wrapper Class for Common API Functions
- Building a Web Portal with Salesforce.com Data
- Add and Remove Tags on a Single Record
- Add and Remove Tags on Multiple Records
- Updating Tag Definitions

As you become more experienced developing on the platform, you'll find that there are some types of applications and integrations that can't be handled by modifying a single app. For situations like these, you can leverage the powerful, SOAP-based API to write client applications that are created and executed outside of Salesforce.com.

In this chapter, you'll learn about choosing a development language, selecting a WSDL document, and managing API authentication, sessions, and timeouts. We'll also take a look at what it takes to build a full-fledged client application that demonstrates several API best practices.

# Setting Up Your Salesforce.com Web Services API Applications

Before working with the API in the recipes in this chapter, perform the following setup steps.

1. Select a Development Language on page 228
2. Create an Integration User on page 228
3. Select a WSDL on page 229
4. Generate a WSDL Document on page 230
5. If You Use the Partner WSDL on page 231
6. Log In to and Out of the API on page 233
7. Manage Sessions on page 236
8. Change the Session Timeout Value on page 237

## Select a Development Language

Choose the programming language or languages in which you'll write your application.

If you're interested in building a Web control or client application, write your code in any language that supports Web services, including Java, Perl, Python, PHP, Ruby on Rails, C#.NET, Visual Basic.NET, and Cocoa for Mac OS X. You can find toolkits and code samples for several Web-services-enabled languages on the Developer Force website at [wiki.developerforce.com/index.php/API](http://wiki.developerforce.com/index.php/API).

## Create an Integration User

Client applications that access Salesforce.com through the API must first log in as a Salesforce.com user for authentication. Create a special user in your organization, solely for integration purposes. That way, even if an actual user leaves your organization, you'll always have a user with the correct permissions available.

Assign this user a special profile with the following permissions selected:

- “API Only”—Specifies that the user can only log in through the API. This prevents the user from being used for any purpose other than integration scenarios.
- “Modify All Data”—Specifies that the user can view any data stored in the database and edit any field with the editable flag. (Some fields, like `CreatedDate`, do not have the editable flag set and cannot be edited by any user, regardless of the “Modify All Data”

permission.) This permission is also required for any user who wants to upsert non-unique external IDs through the API.



**Tip:** If you don't need to worry about external IDs, use the “Modify All Records” permission for a more secure integration. This permission further restricts the integration user's access.

In addition, consider restricting the following to enable a more secure integration:

- All logins use secure access (HTTPS).
- The integration user's access to just those objects required for the integration.
- The IP addresses that the integration user can use, perhaps to just the IP address of the server.
- The organization-wide sharing model—select the lowest level of the hierarchy for the integration user to make changes.
- All passwords are considered strong and contain at least 20 random characters.

## Select a WSDL

A WSDL document is an XML file that describes the format of messages you send and receive from a Web service. It's the protocol that your development environment's SOAP client uses to communicate with external services like Salesforce.com.

Salesforce.com provides two primary WSDL documents for operating on objects and fields in an organization, plus three additional WSDL documents for specific features of workflow rules and the API. Choose the WSDL document you should download and consume based on the type of application you're going to develop:

### Enterprise WSDL

The enterprise WSDL is a strongly typed WSDL document for customers who want to build an integration with their Salesforce.com organization only, or for partners who are using tools like Tibco or webMethods to build integrations that require strong typecasting.

Strong typing means that an object in Salesforce.com has an equivalent object in Java, .NET, or whatever environment is accessing the API. This model generally makes it easier to write code because you don't need to deal with any underlying XML structures. It's also safer because data and schema dependencies are resolved at compile time, not at runtime.

The downside of the enterprise WSDL, however, is that it only works with the schema of a single Salesforce.com organization because it's bound to all of the unique objects and fields that exist in that organization's data model. Consequently, if you use the enterprise WSDL, you must download and re-consume it whenever your organization makes a change to its

custom objects or fields. Additionally, you can't use it to create solutions that can work for multiple organizations.

### Partner WSDL

The partner WSDL is a loosely typed WSDL document for customers, partners, and ISVs who want to build an integration or an AppExchange app that can work across multiple Salesforce.com organizations.

With this WSDL document, the developer is responsible for marshaling data in the correct object representation, which typically involves editing the XML. However, you're also freed from being dependent on any particular data model or Salesforce.com organization. Consequently, if you use the partner WSDL, you only need to download and consume it once, regardless of any changes to custom objects or fields.

### Outbound Message WSDL

The Outbound Message WSDL document is for developers who want to send outbound messages from a workflow rule or approval process to an external service.

### Apex WSDL

The Apex WSDL document is for developers who want to run or compile Apex scripts in another environment.

### Metadata WSDL

The Metadata WSDL document is for users who want to use the Metadata API to retrieve or deploy customization information, such as custom object definitions and page layouts. See “Understanding the Metadata API” in the *Force.com Metadata API Developer’s Guide*.

### Delegated Authentication WSDL

The delegated authentication WSDL document is for users who want to create a delegated authentication application to support single-sign on. You can also download a client certificate for validating requests generated by Salesforce.com.

## Generate a WSDL Document

If you want to generate a WSDL document other than an Outbound Message WSDL document, log in to your Salesforce.com organization and click **Setup** ▶ **Develop** ▶ **API**. Right-click the WSDL document you want to generate, and select **Save Link As** in Firefox, or **Save Target As** in Internet Explorer.

If you want to view a WSDL document without downloading it, simply click the download link for the WSDL document you want to view.



**Tip:** To keep track of the WSDL documents you download, name them with a date/time stamp.

If you want to generate an Outbound Message WSDL document, click **Setup > Create > Workflow & Approvals > Outbound Messages**. Select the name of the outbound message and then click **Click for WSDL**. This file is bound to the outbound message and contains the instructions about how to reach the endpoint service and what data is sent to it.

## If You Use the Partner WSDL

If you want to use the partner WSDL, but you don't know how to work with the loosely typed SOAP messages, use the following information.

The partner WSDL is based on a generic SObject, which represents a Salesforce.com record such as a particular account or contact. Every SObject has the following properties:

| Name         | Type                                                       | Description                                                                                                                                                                                                                                                                                                                                          |
|--------------|------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type         | string                                                     | The API name of the object on which this SObject is based. For example, Account and Position__c.                                                                                                                                                                                                                                                     |
| ID           | ID                                                         | The unique ID for this SObject. For the <code>create()</code> call, this value is null. For all other API calls, this value must be specified.                                                                                                                                                                                                       |
| Any          | XMLElement[]<br>(in .NET)<br>MessageElement[]<br>(in Java) | An array of fields for the SObject. Each element of the array consists of an XML tag, where the name of the field is the name of the element, and the value of the field is the body of the tag. For example:<br><br><name>value</name>                                                                                                              |
| FieldsToNull | string[]                                                   | An array of one or more field names whose value you want to explicitly set to null. This array is used only with the <code>update()</code> or <code>upsert()</code> calls.<br><br>Note that you can only specify fields that you can update and that are nillable. For example, specifying an ID field or required field results in a runtime error. |

The partner WSDL provides methods that allow you to work with these properties so that you can perform the same tasks with the partner WSDL as you can with the enterprise WSDL. For example, the following Java code creates a job application record using the enterprise WSDL:

```
public Job_Application__c createJobApp(String candidateId,
   String positionId) {
    Job_Application__c jobApp = new Job_Application__c();
    jobApp.setCandidate__c(new ID(candidateId));
    jobApp.setPosition__c(new ID(positionId));
    jobApp.setStatus__c("New");
    SaveResult [] sr = binding.create(new SObject[] {jobApp});
    if(!sr[0].isSuccess())
        throw new SaveException(sr[0]);
    jobApp.setId(sr[0].getId());
    return jobApp;
}
```

The same `createJobApp()` method can also be written in Java with the partner WSDL:

```
public SObject createJobApp(String candidateId,
                            String positionId) {
    SObject jobApp = new SObject();
    // Submit four fields as part of the Any array on the 184
    // Chapter 11:Writing Web Controls and Client Applications
    // Job_Application__c record
    MessageElement[] fields = new MessageElement[3];
    // Candidate id
    field[0] = util.createNewXmlElement("Candidate__c", candidateId);

    // Position id
    field[1] = util.createNewXmlElement("Position__c", positionId);
    // Status
    field[2] = util.createNewXmlElement("Status__c", "New");
    jobApp.set_any(fields);
    jobApp.setTType("Job_Application__c");
    SaveResult [] sr = binding.update(new SObject[] {jobApp});
    if(!sr[0].isSuccess())
        throw new SaveException(sr[0]);
    jobApp.setId(sr[0].getId());
    return jobApp;
}
```

The following VB.NET code creates a position record using the enterprise WSDL:

```
Dim p As New Position__c
p.Id = "a00D0000005iYiq"
p.Name = "Analyst"
p.Status__c = "Open"
binding.create(New sObject() {p})
```

This code can be written using the partner WSDL as follows:

```
Dim p as New SObject
p.Type = "Position__c"
p.Id = "a00D0000005iYiq"
Dim doc as New XmlDocument
Dim e1, e2 as XmlElement
e1 = doc.CreateNewElement("Name")
e2 = doc.CreateNewElement("Status")
e1.InnerText = "Analyst"
e2.InnerText = "Open"
p.Any = new XElement() {e1,e2}
binding.update(New sObject() {p})
```



**Note:** In these examples, notice that Java and .NET use different elements to represent field name/value pairs. For example, given the following name/value pair:

```
<City__c>Chicago</City__c>
```

- Java uses a `MessageElement` where:
  - `City__c` is the `Name`
  - `Chicago` is the `Value`
- .NET uses an `XMLElement` where:
  - `City__c` is the `LocalName`
  - `Chicago` is the `InnerText`

Use the partner WSDL in conjunction with the `describeGlobal()` and `describeSObjects()` API calls to get object metadata. For example, a particular object's type is defined in the `name` field in the returned `DescribeSObjectResult`. Likewise, the name of an object's field is defined in the `name` field of the `Field` type in the returned `DescribeSObjectResult`.

## Log In to and Out of the API

If a client application originates from outside the Salesforce.com user interface, it must first log in to the API. The best practice is to log out of the API when the client application completes its work. Use the following information to understand how to log in and log out from a client application.

Similar to the way the login page works in the Salesforce.com user interface, the `login()` call takes a username and password and executes a login sequence on <https://login.salesforce.com/>. If the login is successful, the `login()` call returns

a session ID and URL. The session ID represents the user's authentication token and the URL points to the host that contains data for the user's organization.



**Note:** For performance and reliability, the platform runs on multiple instances (for example, na1.salesforce.com and na2.salesforce.com), but data for any single organization is always consolidated on a single instance. As long as you use the URL that is returned from the `login()` call, you should never need to know the actual instance that hosts an organization's data.



**Figure 21: Authenticating with the `login()` Call**

Once you've obtained a session ID and server URL, you'll generally include the session ID in every API call, and you'll direct your client to make the API request to the host that you obtained.



**Tip:** It's not necessary to use `login()` when writing an s-control that executes within the Salesforce.com user interface because the user accessing the s-control has already logged in and acquired a session ID.

To log in, acquire a Salesforce.com session ID and the appropriate host for your organization by using the `login()` call.

For example, the following Java code from the wrapper class described in [Using a Wrapper Class for Common API Functions](#) on page 242:

- Logs in to Salesforce.com
- Sets the login time
- Resets the URL for the SOAP binding stub to the returned server URL
- Creates a new session header for the binding class variable
- Updates the wrapper class' `sessionId` and `serverURL` variables

```

/**
 * This method is used to log in to salesforce and set the
 * private class variables for the wrapper, including the
 * session ID.
 */
public void login() throws UnexpectedErrorFault, InvalidIdFault,
  
```

```

        LoginFault, RemoteException,
        ServiceException {

    resetBindingStub();
    LoginResult loginResult = binding.login(username, password);
    this.nextLoginTime = System.currentTimeMillis() +
        (this.sessionlength * 60000);

    this.binding._setProperty(SoapBindingStub.
        ENDPOINT_ADDRESS_PROPERTY,
        loginResult.getServerUrl());
    this.sessionId = loginResult.getSessionId();
    this.serverUrl = loginResult.getServerUrl();

    // Create a new session header object and set the
    // session id to that returned by the login
    SessionHeader sh = new SessionHeader();
    sh.setSessionId(loginResult.getSessionId());
    this.binding.setHeader(new
        SforceServiceLocator().getServiceName().getNamespaceURI(),
        "SessionHeader", sh);
}

```

This VB .NET code performs the same logic as for the VB.NET version of the wrapper class (see [Using a Wrapper Class for Common API Functions](#) on page 242):

```

Public Sub Login()
    Dim lr As sforce.LoginResult
    Me._binding.Url = Me._host
    lr = Me._binding.login(username, password)
    Me._nextLoginTime = Now().AddMinutes(Me.sessionlength)
    'Reset the SOAP endpoint to the returned server URL
    Me._binding.Url = lr.serverUrl
    Me._binding.SessionHeaderValue = New sforce.SessionHeader
    Me._binding.SessionHeaderValue.sessionId = lr.sessionId
    Me._sessionId = lr.sessionId
    Me._serverURL = lr.serverUrl
End Sub

```

To log out of the session you created, issue the `logout()` call.

Java:

```

/**
 * This method is used to log out of salesforce
 */
public void logout() {
    try {
        binding.logout();
    }
    catch (Exception e) {
        System.out.println("Unexpected error:\n\n" + e.getMessage());
    }
}

```

```

    }
}
```

VB.NET:

```

Public Sub Logout()
    Me._binding.logout()
End Sub
```

## Manage Sessions

An integration may last longer than the session timeout value specified for an organization, but logging in to Salesforce.com every time you need to make an API call is inefficient. To manage sessions, use the information in this section.

The session timeout value is the amount of time a single session ID remains valid before expiring. Sessions expire automatically after a predetermined length of inactivity, which can be configured in Salesforce.com by clicking **Setup > Security Controls**. The default is 120 minutes (two hours). If you make an API call, the inactivity timer is reset to zero.

You can manage sessions by writing a method that checks to see whether your session ID is about to expire by comparing your last login time with the current session length. When this method returns true, log in again.

For example, the following Java code from the wrapper class discussed in [Using a Wrapper Class for Common API Functions](#) implements a `loginRequired()` method:

```

/**
 * This method returns true if a login to Salesforce is
 * necessary, otherwise false. It should be used to check the
 * session length before performing any API calls.
 */
private boolean loginRequired() {
    if (sessionId == null || sessionId.length() == 0)
        return true;
    return !isConnected();
}

/**
 * This method checks whether the session is active or not
 * @return boolean
 */
public boolean isConnected() {
    return System.currentTimeMillis() < nextLoginTime;
}
```

This VB.NET function implements the same logic:

```
Private Function loginRequired() As Boolean
    loginRequired = Not (isConnected())
End Function

Public Function isConnected() As Boolean
    If _sessionId <> "" And _sessionId <> Nothing Then
        If Now() > Me._nextLoginTime Then
            isConnected = False
        End If
        isConnected = True
    Else
        isConnected = False
    End If
End Function
```



**Tip:** Be sure that the value you use for session length is no more than the configured session timeout value. Because the session timeout value for an organization is not accessible through the API, it's a good idea to build applications that assume a thirty-minute session timeout so that administrators don't inadvertently break your integrations.

This example is very simple. Another, more robust option is to catch the session expiration remove exception (Exception Code - `INVALID_SESSION_ID`) and only then log in again. This ensures that you only log in when absolutely necessary and that you'll get a new session ID if your current session ever becomes invalid. This method is usually coupled with implementing retry logic.

## Change the Session Timeout Value

To change the session timeout value from the two hour default, so that your integrations can work longer without having to get a new session ID:

1. Log in to the application as an administrator.
2. Click **Setup > Security Controls > Session Settings**.
3. Change the **Timeout** value to one of the few preset values. They range from as little as 15 minutes to as long as 8 hours.



### Note:

- Make sure you update any integration code so that it uses the new timeout value. Otherwise, your integrations might break.
- Changing the session timeout value affects all users in an organization.

## Implementing the query() /queryMore() Pattern

### Problem

You need to issue queries that return more than 2,000 records, but the `query()` call can only return up to 2,000 at a time.

### Solution

Use `queryMore()` to retrieve any additional records in batches of up to 2,000 at a time. The `queryMore()` call takes a single `queryLocator` parameter that specifies the index of the last result record that was returned. This `queryLocator` is created and returned by the previous `query()` or `queryMore()` call.

When the `query()` or `queryMore()` calls return a result with the `isDone` flag set to true, there are no more records to process.

For example, the following Java code implements the `query()`/`queryMore()` pattern when querying leads:

```
QueryResult queryResult = this.stub.query("Select name From lead");
do {
    for(sObject lead : queryResult.getRecords()) {
        System.out.println(lead.get_any()[0].getValue());
    }
    if(queryResult.isDone())
        break;
    queryResult = this.stub.queryMore(queryResult.
                                      getQueryLocator());
} while(true);
```

This code implements `query()`/`queryMore()` in VB.NET:

```
Dim lead As sforce.sObject
Dim i As Integer
Dim qr As sforce.QueryResult = binding.query("select name from lead")
Do
    For i = 0 To qr.records.Length
        lead = qr.records(i)
        Console.WriteLine(lead.Any(0).InnerText)
    Next
    If qr.done Then Exit Do
    qr = binding.queryMore(qr.queryLocator)
Loop
```

The `query()`/`queryMore()` batch size defaults to 500 records, but can be as small as 200 or as large as 2000. To change the batch size, use the `QueryOptions` header.

**Note:**

- If you use `query()`/`queryMore()` during a long-running integration scenario where you need to log in again to get new session IDs, the `queryLocator` cursor remains valid after you log in, as long as you get the next batch of records within fifteen minutes of idle time.
- Only 10 `queryLocator` cursors can be active for an organization at any one time.

**See Also**

- [Manage Sessions](#) on page 236
- [Using a Wrapper Class for Common API Functions](#) on page 242
- [Batching Records for API Calls](#) on page 239

## Batching Records for API Calls

**Problem**

You want to create, update, or delete records in Salesforce.com, but you have more than 200 records you want to process, which exceeds the maximum allowed per call.

**Solution**

Write a method that batches the records into multiple API calls.

For example, the following Java code from the wrapper class described in [Using a Wrapper Class for Common API Functions](#) implements a `create()` method that takes an array of SObjects and a batch size as parameters. Any method that calls `create()` can pass in any number of records and dynamically vary the batch size to improve performance:

```
/**
 * This method creates an array of sObjects with a specified
 * batchSize.
 * @param records
 * @param batchSize
 * @return SaveResult[]
 */
public SaveResult[] create(SObject[] records, int batchSize)
    throws InvalidSObjectFault, UnexpectedErrorFault,
        InvalidIdFault, RemoteException,
        ServiceException {
    if (batchSize > 200 || batchSize < 1)
        throw new IllegalArgumentException(
            "batchSize must be between 1 and 200");
    return batch(records, batchSize, new CreateBatcher());
```

```

}

private SaveResult[] batch(SObject[] records, int batchSize,
                         Batcher batchOperation)
    throws UnexpectedErrorFault, InvalidIdFault,
           LoginFault, RemoteException, ServiceException {
    if (records.length <= batchSize) {
        checkLogin();
        return batchOperation.perform(records);
    }
    SaveResult[] saveResults = new SaveResult[records.length];
    SObject[] thisBatch = null;
    int pos = 0;
    while (pos < records.length) {
        int thisBatchSize = Math.min(batchSize,
                                      records.length - pos);
        if (thisBatch == null ||
            thisBatch.length != thisBatchSize)
            thisBatch = new SObject[thisBatchSize];

        System.arraycopy(records, pos, thisBatch, 0,
                        thisBatchSize);
        SaveResult [] batchResults = batch(thisBatch,
   thisBatchSize,
   batchOperation);
        System.arraycopy(batchResults, 0, saveResults,
                        pos, thisBatchSize);
        pos += thisBatchSize;
    }
    return saveResults;
}

private abstract class Batcher {
    abstract SaveResult[] perform(SObject [] records)
        throws UnexpectedErrorFault, InvalidIdFault,
               LoginFault, RemoteException,
               ServiceException;
}

private class CreateBatcher extends Batcher {
    SaveResult [] perform(SObject [] records)
        throws UnexpectedErrorFault, InvalidIdFault,
               LoginFault, RemoteException, ServiceException {
        checkLogin();
        return binding.create(records);
    }
}

private class UpdateBatcher extends Batcher {
    SaveResult [] perform(SObject [] records)
        throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
               RemoteException, ServiceException {
        checkLogin();
    }
}

```

```

        return binding.update(records);
    }
}

```

This VB.NET function implements the same logic:

```

Public Function create(ByVal records() As sObject,
                      Optional ByVal batchSize As Integer = 200)

    As sforce.SaveResult()
    Return batch(records, batchSize, New CreateBatcher)
End Function

Private Function batch(ByVal records() As sObject,
                      ByVal batchSize As Integer,
                      ByVal oper As Batcher)
    As sforce.SaveResult()
If (records.Length <= batchSize) Then
    batch = oper.perform(Binding, records)
    Exit Function
End If

Dim saveResults(records.Length - 1) As sforce.SaveResult
Dim thisBatch As sforce.sObject()
Dim pos As Integer = 0
Dim thisBatchSize As Integer

While (pos < records.Length)
    thisBatchSize = Math.Min(batchSize,
                           records.Length - pos)
    ReDim thisBatch(thisBatchSize)
    System.Array.Copy(records, pos, thisBatch,
                      0, thisBatchSize)
    Dim sr As sforce.SaveResult() =
        oper.perform(Binding, thisBatch)
    System.Array.Copy(sr, 0, saveResults, pos, thisBatchSize)

    pos += sr.Length
End While
batch = saveResults
End Function

Private Class Batcher
    Public Function perform(ByVal binding As sforce.SforceService,
                           ByVal records As sforce.sObject())
        As sforce.SaveResult()
        perform = Nothing
    End Function
End Class

Private Class CreateBatcher
    Inherits Batcher
    Public Overloads Function perform(

```

```
        ByVal binding As sforce.SforceService,
        ByVal records As sforce.sObject())
    As sforce.SaveResult()
    perform = binding.create(records)
End Function
End Class

Private Class UpdateBatcher
Inherits Batcher
Public Overloads Function perform(
    ByVal binding As sforce.SforceService,
    ByVal records As sforce.sObject())
    As sforce.SaveResult()
    perform = binding.update(records)
End Function
End Class
```

## See Also

- [Manage Sessions](#) on page 236
- [Using a Wrapper Class for Common API Functions](#) on page 242
- [Log In to and Out of the API](#) on page 233

# Using a Wrapper Class for Common API Functions

## Problem

You find yourself writing similar sections of code wherever you need to make calls to the API in a client application.

## Solution

Use an API wrapper class to abstract common functions whenever you write client applications and integrations. A wrapper class makes your integration more straightforward to develop and maintain, keeps the logic necessary to make API calls in one place, and affords easy reuse across all components that require API access.

Wrapper classes typically include methods for the following types of actions:

- Logging in
- Managing sessions
- Querying with the `query()`/`queryMore()` pattern
- Batching records for create, update, delete, and so on

For example, the following Java code is a complete implementation of the wrapper class used in [Building a Web Portal with Salesforce.com Data](#) on page 255:

```
package com.sforce.client;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import org.apache.axis.transport.http.HTTPConstants;

import com.sforce.soap.partner.AssignmentRuleHeader;
import com.sforce.soap.partner.LoginResult;
import com.sforce.soap.partner.QueryOptions;
import com.sforce.soap.partner.QueryResult;
import com.sforce.soap.partner.SaveResult;
import com.sforce.soap.partner.SessionHeader;
import com.sforce.soap.partner.SforceServiceLocator;
import com.sforce.soap.partner.SoapBindingStub;
import com.sforce.soap.partner.fault.InvalidIdFault;
import com.sforce.soap.partner.fault.InvalidSObjectFault;
import com.sforce.soap.partner.fault.LoginFault;
import com.sforce.soap.partner.fault.UnexpectedErrorFault;
import com.sforce.soap.partner.sobject.SObject;

public class Client {

    // Private wrapper class variables
    private String username;
    private String password;
    private URL host;
    private int querySize;
    private int sessionlength;
    private String sessionId;
    private String serverUrl;
    private long nextLoginTime;
    private SoapBindingStub binding;
    private boolean useCompression;

    private QueryOptions queryOptions;
    private AssignmentRuleHeader assignmentRules;

    /**
     * This method initializes the private class variables
     */
    public Client() throws MalformedURLException {
        this.querySize = 500;
        this.sessionlength = 29;
        this.useCompression = true;
        this.host = new URL(
            "https://www.salesforce.com/services/Soap/u/18.0");
    }
}
```

```
}

/**
 * These methods get and set the private class variables
 */
public String getUsername() {
    return this.username;
}

public void setUsername(String value) {
    this.username = value;
}

public String getPassword() {
    return this.password;
}

public void setPassword(String value) {
    this.password = value;
}

public URL getHost() {
    return this.host;
}

public void setHost(URL value) {
    this.host = value;
}

public void setHost(String url) throws MalformedURLException {
    this.host = new URL(url);
}

public String getServerURL() {
    return this.serverUrl;
}

public int getQuerySize() {
    return this.querySize;
}

public void setQuerySize(int value) {
    this.querySize = value;
}

public int getSessionlength() {
    return this.sessionlength;
}

public void setSessionlength(int value) {
    this.sessionlength = value;
}

public boolean getUseCompression() {
    return this.useCompression;
}
```

```

public void setUseCompression(boolean value) {
    this.useCompression = value;
    setCompressionOnBinding();
}

< /**
 * This method is used to log in to salesforce and set the
 * private class variables for the wrapper, including the
 * session ID.
 */
public void login() throws UnexpectedErrorFault, InvalidIdFault,
                           LoginFault, RemoteException,
                           ServiceException {

    resetBindingStub();
    LoginResult loginResult = binding.login(username, password);
    this.nextLoginTime = System.currentTimeMillis() +
        (this.sessionlength * 60000);

    this.binding._setProperty(SoapBindingStub.
        ENDPOINT_ADDRESS_PROPERTY,
        loginResult.getServerUrl());
    this.sessionId = loginResult.getSessionId();
    this.serverUrl = loginResult.getServerUrl();

    // Create a new session header object and set the
    // session id to that returned by the login
    SessionHeader sh = new SessionHeader();
    sh.setSessionId(loginResult.getSessionId());
    this.binding.setHeader(new SforceServiceLocator().
        getServiceName().getNamespaceURI(),
        "SessionHeader", sh);
}

private void checkLogin() throws UnexpectedErrorFault,
                               InvalidIdFault, LoginFault, RemoteException,
                               ServiceException {
    if (this.loginRequired())
        login();
}

< /**
 * This method is used to log in with an existing sessionId
 * @param String sid sessionId
 * @param String sURL serverUrl
 */
public void loginBySessionId(String sid, String sURL)
    throws ServiceException {
    this.nextLoginTime = System.currentTimeMillis() +
        (this.sessionlength * 60000);
    resetBindingStub();
    binding._setProperty(

```

```
        SoapBindingStub.ENDPOINT_ADDRESS_PROPERTY, sURL);
this.sessionId = sid;
this.serverUrl = sURL;
SessionHeader sh = new SessionHeader();
sh.setSessionId(sid);
binding.setHeader(new SforceServiceLocator().
    getServiceName().getNamespaceURI(),
    "SessionHeader", sh);
}

/** This method resets the binding object back to its
 * initial state.
 */
private void resetBindingStub() throws ServiceException {
    this.binding = (SoapBindingStub) new
        SforceServiceLocator().getSoap(this.host);
    this.binding.setTimeout(60000);
    setCompressionOnBinding();
    this.assignmentRules = null;
    this.queryOptions = null;
}

private void setCompressionOnBinding() {
    binding._setProperty(HTTPConstants.MC_ACCEPT_GZIP,
        useCompression);
    binding._setProperty(HTTPConstants.MC_GZIP_REQUEST,
        useCompression);
}

/**
 * This method checks whether the session is active or not
 * @return boolean
 */
public boolean isConnected() {
    return System.currentTimeMillis() < nextLoginTime;
}

/**
 * This method returns true if a login to Salesforce is
 * necessary, otherwise false. It should be used to check the
 * session length before performing any API calls.
 */
private boolean loginRequired() {
    if (sessionId == null || sessionId.length() == 0)
        return true;
    return !isConnected();
}

private void setBatchSizeHeader(int batchSize) {
    if (queryOptions == null) {
        this.queryOptions = new QueryOptions();
        binding.setHeader(new SforceServiceLocator().
            getServiceName().getNamespaceURI(),
```

```
        "QueryOptions", queryOptions);
    }
    queryOptions.setBatchSize(batchSize);
}

/**
 * This method queries the database and returns the results.
 * @param String strSOQLStmt
 * @return SObject[]
 */
public QueryResult executeQuery(String strSOQLStmt,
                                Integer queryBatchSize)
    throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
           RemoteException, ServiceException {

    checkLogin();
    setBatchSizeHeader(queryBatchSize ==
                       null ? querySize : queryBatchSize);
    return binding.query(strSOQLStmt);
}

public QueryResult executeSOQL(String strSOQLStmt)
    throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
           RemoteException, ServiceException {
    return executeQuery(strSOQLStmt, null);
}

public QueryResult executeQueryMore(String queryLocator)
    throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
           RemoteException, ServiceException {
    checkLogin();
    return binding.queryMore(queryLocator);
}

/**
 * This method sets the assignment rule header.
 * @param ruleId
 */
public void setAssignmentRuleHeaderId(String ruleId) {
    setAssignmentRuleHeader(ruleId, false);
}

/**
 * This method sets the assignment rule header with a
 * default ruleId.
 * @param ruleId
 */
public void setAssignmentRuleHeaderToDefault
    (boolean runDefaultRule) {
    setAssignmentRuleHeader(null, runDefaultRule);
}

private void setAssignmentRuleHeader(String ruleId,
```

```

        boolean useDefault) {
    if (this.assignmentRules == null) {
        this.assignmentRules = new AssignmentRuleHeader();
        binding.setHeader(new SforceServiceLocator().
            getServiceName().getNamespaceURI(),
            "AssignmentRuleHeader", this.assignmentRules);
    }
    this.assignmentRules.setUseDefaultRule(useDefault);
    this.assignmentRules.setAssignmentRuleId(ruleId);
}

public SoapBindingStub getBinding()
    throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
        RemoteException, ServiceException {
    checkLogin();
    return this.binding;
}

/**
 * This method creates an array of sObjects with a specified
 * batchSize.
 * @param records
 * @param batchSize
 * @return SaveResult[]
 */
public SaveResult[] create(SObject[] records, int batchSize)
    throws InvalidSObjectFault, UnexpectedErrorFault,
        InvalidIdFault, RemoteException,
        ServiceException {
    if (batchSize > 200 || batchSize < 1)
        throw new IllegalArgumentException(
            "batchSize must be between 1 and 200");
    return batch(records, batchSize, new CreateBatcher());
}

public SaveResult[] create(SObject[] records)
    throws InvalidSObjectFault, UnexpectedErrorFault,
        InvalidIdFault, RemoteException, ServiceException {

    return create(records, 200);
}

public SaveResult[] update(SObject[] records, int batchSize)
    throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
        RemoteException, ServiceException {
    if (batchSize > 200 || batchSize < 1)
        throw new IllegalArgumentException(
            "batchSize must be between 1 and 200");

    return batch(records, batchSize, new UpdateBatcher());
}

/**
 * This method updates an array of sObjects with a specified

```

```

    * batchSize.
    * @param records
    * @param batchSize
    * @return SaveResult[]
 */
public SaveResult[] update(SObject[] records)
    throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
           RemoteException, ServiceException {
    return update(records, 200);
}

private SaveResult[] batch(SObject[] records, int batchSize,
                         Batcher batchOperation)
    throws UnexpectedErrorFault, InvalidIdFault,
           LoginFault, RemoteException, ServiceException {
    if (records.length <= batchSize) {
        checkLogin();
        return batchOperation.perform(records);
    }
    SaveResult[] saveResults = new SaveResult[records.length];
    SObject[] thisBatch = null;
    int pos = 0;
    while (pos < records.length) {
        int thisBatchSize = Math.min(batchSize,
                                     records.length - pos);
        if (thisBatch == null ||
            thisBatch.length != thisBatchSize)
            thisBatch = new SObject[thisBatchSize];

        System.arraycopy(records, pos, thisBatch, 0,
                        thisBatchSize);
        SaveResult [] batchResults = batch(thisBatch,
   thisBatchSize,
   batchOperation);
        System.arraycopy(batchResults, 0, saveResults,
                        pos, thisBatchSize);
        pos += thisBatchSize;
    }
    return saveResults;
}

private abstract class Batcher {
    abstract SaveResult[] perform(SObject [] records)
        throws UnexpectedErrorFault, InvalidIdFault,
               LoginFault, RemoteException,
               ServiceException;
}

private class CreateBatcher extends Batcher {
    SaveResult [] perform(SObject [] records)
        throws UnexpectedErrorFault, InvalidIdFault,
               LoginFault, RemoteException, ServiceException {
        checkLogin();
        return binding.create(records);
}

```

```

        }
    }

    private class UpdateBatcher extends Batcher {
        SaveResult [] perform(SObject [] records)
            throws UnexpectedErrorFault, InvalidIdFault, LoginFault,
                   RemoteException, ServiceException {
            checkLogin();
            return binding.update(records);
        }
    }
}

```

This code implements the VB.NET version of the wrapper class used in [Building a Web Portal with Salesforce.com Data](#) on page 255:

```

Imports Microsoft.VisualBasic
Imports System
Imports System.Collections
Imports sforce

Public Class Client

    Private _binding As SforceServiceCompressed
    Private _username As String
    Private _password As String
    Private _host As String
    Private _querySize As Integer
    Private _sessionlength As Integer
    Private _sessionId As String
    Private _serverURL As String
    Private _nextLoginTime As DateTime

    'Initialize private variables for the class
    Sub New()
        Me._binding = New SforceServiceCompressed
        Me._querySize = 500
        Me._sessionlength = 29
        Me._host = "https://www.salesforce.com/services/Soap/u/10.0"
    End Sub

    'Expose variables to calling function
    Public Property username() As String
        'Allows calling class to get values
        Get
            Return Me._username
        End Get
        'Allows calling class to set value
        Set(ByVal Value As String)
            Me._username = Value
        End Set
    End Property

    Public Property password() As String

```

```

        Get
            Return Me._password
        End Get
        Set(ByVal Value As String)
            Me._password = Value
        End Set
    End Property

    Public Property host() As String
        Get
            Return Me._host
        End Get
        Set(ByVal Value As String)
            Me._host = Value
        End Set
    End Property

    Public ReadOnly Property serverURL() As String
        Get
            Return Me._serverURL
        End Get
    End Property

    Public Property querySize() As Integer
        Get
            Return Me._querySize
        End Get
        Set(ByVal Value As Integer)
            Me._querySize = Value
        End Set
    End Property

    Public Property sessionlength() As Integer
        Get
            Return Me._sessionlength
        End Get
        Set(ByVal Value As Integer)
            Me._sessionlength = Value
        End Set
    End Property

    'In case of proxy server...
    Public Property proxy() As System.Net.WebProxy
        Get
            Return Me._binding.Proxy
        End Get
        Set(ByVal Value As System.Net.WebProxy)
            Me._binding.Proxy = Value
        End Set
    End Property

    ' Compress SOAP messages
    Public Property useCompression() As Boolean
        Get
            Return Me._binding.AcceptCompressedResponse()
        End Get

```

```

Set(ByVal Value As Boolean)
    Me._binding.AcceptCompressedResponse = Value
    Me._binding.SendCompressedRequest = Value
End Set
End Property

Public Sub Login()
    Dim lr As sforce.LoginResult
    Me.binding.Url = Me.host
    lr = Me._binding.login(username, password)
    Me._nextLoginTime = Now().AddMinutes(Me.sessionlength)
    'Reset the SOAP endpoint to the returned server URL
    Me._binding.Url = lr.serverUrl
    Me._binding.SessionHeaderValue = New sforce.SessionHeader
    Me._binding.SessionHeaderValue.sessionId = lr.sessionId
    Me._sessionId = lr.sessionId
    Me._serverURL = lr.serverUrl
End Sub

Public Sub loginBySessionId(ByVal sid As String,
                           ByVal sURL As String)
    Me._nextLoginTime = Now().AddMinutes(Me.sessionlength)
    Me._binding.Url = sURL
    Me._binding.SessionHeaderValue = New sforce.SessionHeader
    Me._binding.SessionHeaderValue.sessionId = sid
    Me._sessionId = sid
    Me._serverURL = sURL
End Sub

Public Function isConnected() As Boolean
    If _sessionId <> "" And _sessionId <> Nothing Then
        If Now() > Me._nextLoginTime Then
            isConnected = False
        End If
        isConnected = True
    Else
        isConnected = False
    End If
End Function

Private Function loginRequired() As Boolean
    loginRequired = Not (isConnected())
End Function

Public Function executeQuery(ByVal strSOQLStmt As String,
                           Optional ByVal queryBatchSize As Integer = -1)
                           As sforce.QueryResult
    If queryBatchSize = -1 Then
        queryBatchSize = _querySize
    End If

```

```

If (Me.loginRequired()) Then
    Login()
End If
_binding.QueryOptionsValue = New sforce.QueryOptions
_binding.QueryOptionsValue.batchSizeSpecified = True
_binding.QueryOptionsValue.batchSize = queryBatchSize
executeQuery = _binding.query(strSQLStmt)
End Function

Public Function executeQueryMore(ByVal queryLocator As String)
    As sforce.QueryResult
    If loginRequired() Then Login()
    Return _binding.queryMore(queryLocator)
End Function

Public Sub setAssignmentRuleHeaderId(ByVal ruleId As String)
    _binding.AssignmentRuleHeaderValue =
        New AssignmentRuleHeader
    _binding.AssignmentRuleHeaderValue.assignmentRuleId = ruleId
End Sub

Public Sub setAssignmentRuleHeaderToDefault(
    ByVal runDefaultRule As Boolean)

    _binding.AssignmentRuleHeaderValue = New AssignmentRuleHeader
    _binding.AssignmentRuleHeaderValue.useDefaultRule =
        runDefaultRule
End Sub

Public ReadOnly Property Binding() As sforce.SforceService
    Get
        Return _binding
    End Get
End Property

Public Function create(ByVal records() As sObject,
    Optional ByVal batchSize As Integer = 200)

    As sforce.SaveResult()
    Return batch(records, batchSize, New CreateBatcher)
End Function

Public Function update(ByVal records() As sObject,
    Optional ByVal batchSize As Integer = 200)

    As sforce.SaveResult()
    Return batch(records, batchSize, New UpdateBatcher)
End Function

```

```

Private Function batch(ByVal records() As sObject,
                      ByVal batchSize As Integer,
                      ByVal oper As Batcher)
    As sforce.SaveResult()
If (records.Length <= batchSize) Then
    batch = oper.perform(Binding, records)
    Exit Function
End If

Dim saveResults(records.Length - 1) As sforce.SaveResult
Dim thisBatch As sforce.sObject()
Dim pos As Integer = 0
Dim thisBatchSize As Integer

While (pos < records.Length)
    thisBatchSize = Math.Min(batchSize,
                           records.Length - pos)
    ReDim thisBatch(thisBatchSize)
    System.Array.Copy(records, pos, thisBatch,
                      0, thisBatchSize)
    Dim sr As sforce.SaveResult() =
        oper.perform(Binding, thisBatch)
    System.Array.Copy(sr, 0, saveResults, pos, thisBatchSize)

    pos += sr.Length
End While
batch = saveResults
End Function

Private Class Batcher
    Public Function perform(ByVal binding
                           As sforce.SforceService,
                           ByVal records
                           As sforce.sObject())
        As sforce.SaveResult()
        perform = Nothing
    End Function
End Class

Private Class CreateBatcher
    Inherits Batcher
    Public Overloads Function perform(
        ByVal binding As sforce.SforceService,
        ByVal records As sforce.sObject())
        As sforce.SaveResult()
        perform = binding.create(records)
    End Function
End Class

Private Class UpdateBatcher
    Inherits Batcher
    Public Overloads Function perform(
        ByVal binding As sforce.SforceService,
        ByVal records As sforce.sObject())
        As sforce.SaveResult()

```

```
        perform = binding.update(records)
    End Function
End Class
End Class
```

## See Also

- [Building a Web Portal with Salesforce.com Data](#) on page 255
- [Log In to and Out of the API](#) on page 233
- [Manage Sessions](#) on page 236
- [Implementing the query\(\) /queryMore\(\) Pattern](#) on page 238
- [Batching Records for API Calls](#) on page 239

# Building a Web Portal with Salesforce.com Data

## Problem

You want to build a Web portal for the Recruiting app that allows visitors to apply online for open positions. The portal needs to include the following Web pages:

- A list view of all currently open positions, with data from the position records that are stored in Salesforce.com
- Detail views of all currently open positions, also with data from the position records that are stored in Salesforce.com
- An online application form that allows a visitor to apply for an open position. When the user clicks **Submit**, the data is sent back to Salesforce.com as a new job application and candidate record.

Most importantly, your Web portal visitors shouldn't have to log in to view the open positions in your organization.



**Note:** This Web portal is part of the Application Laboratory class offered by salesforce.com Training & Certification. For more information, see [www.salesforce.com/training](http://www.salesforce.com/training).

|             |                                           |                     |
|-------------|-------------------------------------------|---------------------|
| IT          | <a href="#">Sr. Network Engineer</a>      | [ : New York ]      |
| Sales       | <a href="#">Business Analyst</a>          | [ : New York ]      |
| Sales       | <a href="#">Sr. Business Analyst</a>      | [ : New York ]      |
| IT          | <a href="#">Network Analyst</a>           | [ : Chicago ]       |
| Support     | <a href="#">Technical Service Rep</a>     | [ : Chicago ]       |
| IT          | <a href="#">Manager</a>                   | [ : Chicago ]       |
| Finance     | <a href="#">Director</a>                  | [ : New York ]      |
| IT          | <a href="#">Sr. Application Developer</a> | [ : Chicago ]       |
| Engineering | <a href="#">Technical Architect</a>       | [ : San Francisco ] |
| Support     | <a href="#">Manager</a>                   | [ : Chicago ]       |
| Support     | <a href="#">Service Rep</a>               | [ : Chicago ]       |
| IT          | <a href="#">SW Engineer</a>               | [ : San Francisco ] |

**Figure 22: The Web Portal's Job Listings Page**

**Technical Architect**

**Location:** San Francisco

**Job Code:**

**Description:**  
The TA is the leader of our technical teams.  
The TA will innovate and guide us to develop the next generation of our solutions.

**Responsibilities:**  
\*Mentor jr team members  
\*Facilitate meetings  
\*Project Management

**Required Skills/Experience:**  
\*C#  
\*Java  
\*ASP.NET  
\*JSP

[Apply Now](#)

**Figure 23: The Web Portal's Position Detail Page**

**Technical Architect**

**Location:** San Francisco  
**Job Type:** Full-Time  
**Job Code:**

|                                                                                                 |                                                                                             |
|-------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| <b>First Name:</b> <input type="text"/>                                                         | <b>Currently Employed:</b> <input type="checkbox"/>                                         |
| <b>Last Name:</b> <input type="text"/>                                                          | <b>Most Recent Employer:</b> <input type="text"/>                                           |
| <b>Gender:</b> <input type="radio"/> Male<br><input type="radio"/> Female                       | <b>Industry Experience (Years):</b> <input type="text"/>                                    |
| <b>Street:</b> <input type="text"/>                                                             | <b>Highest Education:</b> <input type="text"/> HS Diploma <input checked="" type="button"/> |
| <b>City:</b> <input type="text"/>                                                               | <b>US Citizen:</b> <input type="checkbox"/>                                                 |
| <b>State:</b> <input type="text"/>                                                              | <b>Visa Required:</b> <input type="checkbox"/>                                              |
| <b>Postal Code:</b> <input type="text"/>                                                        |                                                                                             |
| <b>Country:</b> <input type="text"/>                                                            |                                                                                             |
| <b>Email:</b> <input type="text"/>                                                              |                                                                                             |
| <b>Phone:</b> <input type="text"/>                                                              |                                                                                             |
| <b>Mobile Phone:</b> <input type="text"/>                                                       |                                                                                             |
| <b>Fax:</b> <input type="text"/>                                                                |                                                                                             |
| <b>Please attach your resume:</b> <input type="text"/> <input type="button" value="Browse..."/> |                                                                                             |
| <input type="button" value="Submit"/>                                                           |                                                                                             |

**Figure 24: The Web Portal's Application Page**

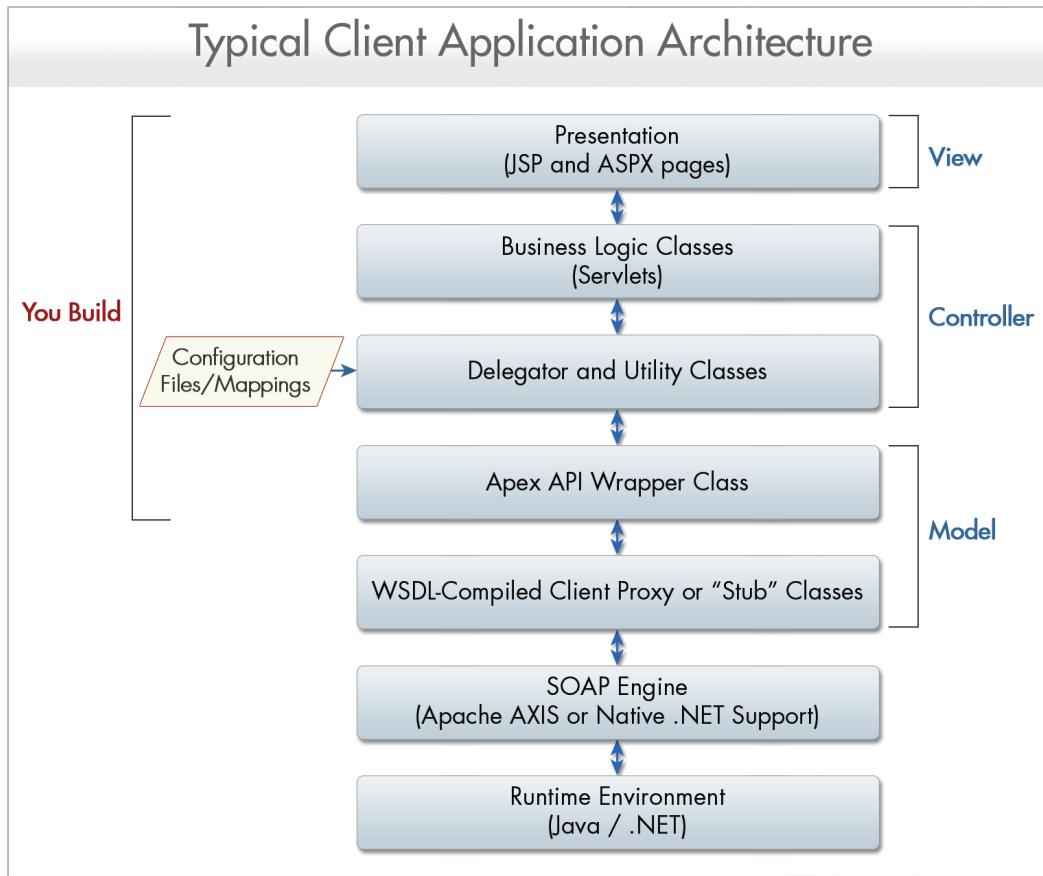
## Solution

Write a client application that runs on an external server and uses the API to access Salesforce.com data.



**Note:** Because there's a lot of code involved with this solution, this recipe discusses how such a client can be designed, and some of the features that can be implemented. To download the complete Java, C#.NET, or VB.NET code that implements this client application, visit [developer.force.com/books/cookbook](http://developer.force.com/books/cookbook).

The following diagram shows the key components of such an application. While there can be some overlap, each component represents a different aspect of the MVC (Model-View-Controller) design paradigm:



**Figure 25: Typical Client Application Architecture**

### Model

The API wrapper class and the WSDL-compiled proxy classes provide non-application-specific access to the data in Salesforce.com. For the wrapper class used to implement the Web portal application, see [Using a Wrapper Class for Common API Functions](#) on page 242.

### View

The JSP or ASPX pages contain the user interface of the application, including how the data is displayed for and captured from the user.

### Controller

The delegator, utility, and business logic classes define the application-specific logic, including the logic that controls how captured data is returned to Salesforce.com as new or updated records.

Of particular note is the delegator class that the Web portal application uses to provide common, reusable code for creating and updating the key objects related to the application, such as Job Application and Candidate. Unlike the API wrapper class, which can be reused by many different client applications, the delegator is application-specific, providing an additional layer of abstraction between the API and the logic required to display the application's pages.

For example, the following Java-based delegator method provides the logic for creating a job application record. Based on the partner WSDL, it prepares a single record and passes it to the wrapper class for creation via the API:

```
public SObject createJobApp(String candidateId,
                           String positionId) {

    SObject application = new SObject();

    try {
        MessageElement[] fields = new MessageElement[3];
        MessageElement field;

        //Candidate id
        field = util.createNewXmlElement("Candidate__c",
   candidateId);
        fields[0] = field;

        //Positionid
        field = util.createNewXmlElement("Position__c",
   positionId);
        fields[1] = field;

        //Status
        field = util.createNewXmlElement("Status__c",
   "New");
        fields[2] = field;

        application.setAny(fields);
        application.setType("Job_Application__c");
        application = createOneRecord(application);

    } catch (Exception e) {
        System.out.println(e.getMessage());
    }

    return application;
}
```

This same logic can be implemented in VB.NET as follows:

```
Public Function createJobApp(ByVal candidateId As String,
                           ByVal positionId As String) As sforce.sObject
    Dim application As New sforce.sObject

    Dim fields(2) As System.Xml.XmlElement
    Dim field As System.Xml.XmlElement
```

```

' Candidate id
field = util.createNewXmlElement("Candidate__c",
                                candidateId)
fields(0) = field

' Position id
field = util.createNewXmlElement("Position__c",
                                positionId)
fields(1) = field

' Status
field = util.createNewXmlElement("Status__c",
                                "New")
fields(2) = field

application.Any = fields
application.type = "Job_Application__c"

application = createOneRecord(application)

Return application
End Function

```

## Discussion

The code that implements this Web portal client application also uses a configuration file and SOAP message compression, two best practices for client application development:

- Using a configuration file to control dynamic aspects of a client application is highly recommended because it reduces code maintenance time. It can include properties such as the API URL, username, password, and any SOQL or SOSL queries that drive business logic. For example, by storing the URL of the targeted Salesforce.com host in a configuration file, changing an integration target from sandbox to production only requires a simple configuration file edit.

The Java-based solution uses a configuration file named `config.properties`, while the VB.NET and C#.NET solutions use configuration files named `web.config`.

- SOAP messages generated by both an API client and the API service can become very large, especially when they include large clobs of data, such as the resume attachment in the Web portal client application. To avoid lengthy transmission times across the Internet, you can configure your SOAP binding to use GZIP compression to reduce the size of SOAP messages by up to 90%. When the API server receives a compressed message, it decompresses the message, processes it, and then recompresses the response before returning it.

The Java-based solution uses the following classes to compress and decompress SOAP messages:

- `GZipWebRequest.java`

- GZipWebResponse.java
- GZIP2WayRequestStream.java
- GZIP2WayRequestWrapper.java
- GZIP2WayResponseStream.java
- GZIP2WayResponseWrapper.java

The VB.NET and C#.NET solutions use these classes:

- GZipWebRequest.vb/GZipWebRequest.cs
- GZipWebResponse.vb/GZipWebResponse.cs
- SforceServiceCompressed.vb/SforceServiceCompressed.cs

For information about SOAP compression in Java, see

[http://wiki.developerforce.com/index.php/Compression\\_with\\_Axis\\_1.3](http://wiki.developerforce.com/index.php/Compression_with_Axis_1.3).

For information about SOAP compression in VB.NET and C#.NET, see

[http://wiki.developerforce.com/index.php/SOAP\\_Compression](http://wiki.developerforce.com/index.php/SOAP_Compression).

## See Also

- [Log In to and Out of the API](#) on page 233
- [Manage Sessions](#) on page 236
- [Implementing the query\(\)/queryMore\(\) Pattern](#) on page 238
- [Batching Records for API Calls](#) on page 239

# Add and Remove Tags on a Single Record

## Problem

You want to remove a public tag that is no longer applicable and add a new public tag for an existing record.

## Solution

To add or remove a tag on a single record, you should first query the record to determine which tags are already present, before continuing with the `create()` and `delete()` operations.

1. To see the tags on a single contact, for example a contact named John Smith, execute the following in the AJAX Toolkit or your own client application:

```
var johnSmithTags = sforce.connection.query(
    "SELECT Name, Id, ItemId "
    +"FROM ContactTag "
    +"WHERE Item.LastName = '\'Smith\' " +
        "AND Item.FirstName = '\'John\'");
```

johnSmithTags contains the following tag names:

- Northwest
- Staff

John Smith has been promoted from Staff to Senior Staff, a tag that does not yet exist.

2. To create the new Senior Staff tag, execute the following in the AJAX Toolkit or your own client application:

```
var SeniorStaffTag = new sforce.SObject('ContactTag');
SeniorStaffTag.ItemId = johnSmithResults.records[0].Id;
SeniorStaffTag.Name = "Senior Staff";
SeniorStaffTag.Type = "Public";
sforce.connection.create([SeniorStaffTag]);
```

3. Run the query in the first step again. johnSmithTags contains the following tag names:
  - Northwest
  - Senior Staff
  - Staff
4. The Staff position is eliminated. To remove the Staff tag, delete its ID:

```
var staffID = sforce.connection.query(
    "SELECT Id FROM ContactTag " +
    "WHERE Name = 'Staff' AND ItemId = '" +
    "johnSmithTags.records[0].ItemId + "')";
sforce.connection.deleteIds([staffID.records.Id]);
```

5. Run the query in the first step again. johnSmithTags contains the following tag names:
  - Northwest
  - Senior Staff

### Discussion

Query to ensure that records have been tagged appropriately.

The API enforces the same limits as the Salesforce.com user interface on the number of tags that you can create.

Across all users, an organization is limited to:

- 1,000 unique public tags
- 50,000 instances of public tags applied to records

- 5,000,000 instances of personal and public tags applied to records

## See Also

- Viewing Tags on page 37
- Viewing Records with Tags on page 38
- “Tagging Limits” in the Salesforce.com Online Help
- “TagDefinition” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/Content/sforce\\_api\\_objects\\_tagdefinition.htm](http://www.salesforce.com/us/developer/docs/api/Content/sforce_api_objects_tagdefinition.htm)
- “create ()” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_create.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_create.htm)
- “delete ()” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_delete.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_delete.htm)
- *The AJAX Toolkit Developer’s Guide* at [www.salesforce.com/us/developer/docs/ajax/index.htm](http://www.salesforce.com/us/developer/docs/ajax/index.htm)

## Add and Remove Tags on Multiple Records

### Problem

You want to remove public tags that are no longer applicable to multiple records of the same object type and replace them with a different tag.

### Solution

To work with multiple records, loop through all records with a particular tag, collect their `ItemIds`, and edit them on an individual basis.

Suppose that all Northwest and Southwest tags need to be consolidated to a single West Coast tag. To remove the tags and replace them:

1. Store the accounts that contain the tags you want to delete. You can execute the following in the AJAX Toolkit or your own client application:

```
var westCoastAccounts = sforce.connection.query(
    "SELECT ItemId, Id " +
    "FROM AccountTag WHERE " +
    "Name = 'Northwest' OR Name = 'Southwest'");
```

2. Create the new West Coast tag. Add the `ItemIds` by looping through the records:

```
var WestCoastTagArray = new Array();
for (var i = 0; i < westCoastAccounts.size; i++) {
    WestCoastTagArray[i] = new sforce.SObject('AccountTag');
```

```

        WestCoastTagArray[i].Name = "West Coast";
        WestCoastTagArray[i].Type = "Public";
        WestCoastTagArray[i].ItemId =
            westCoastAccounts.records[i].ItemId;
    }
sforce.connection.create(WestCoastTagArray);

```

3. Delete all instances of the Northwest and Southwest tags:

```

var IdsToDelete = new Array();
for (var k = 0; k < westCoastAccounts.size; k++)
{
    IdsToDelete[k] = westCoastAccounts.records[k].Id;
}
sforce.connection.deleteIds(IdsToDelete);

```

## Discussion

Make as few calls to Salesforce.com as possible for the best performance. It is efficient to create a single array, populate it with a `for` loop, and call a single operation, such as `create()` or `delete()`, on that array.

The API enforces the same limits as the Salesforce.com user interface on the number of tags that you can create.

Across all users, an organization is limited to:

- 1,000 unique public tags
- 50,000 instances of public tags applied to records
- 5,000,000 instances of personal and public tags applied to records

## See Also

- [Viewing Tags](#) on page 37
- [Viewing Records with Tags](#) on page 38
- “TagDefinition” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_objects\\_tagdefinition.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_objects_tagdefinition.htm)
- “create()” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_create.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_create.htm)
- “delete()” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_delete.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_delete.htm)

# Updating Tag Definitions

## Problem

You want to correct some public tags that have misspellings in them and remove some that are no longer useful in your application.

## Solution

Run a query to determine the ID of the incorrect tag, then rename it. For example, suppose you see that a user has created a tag called WC. You want to retrieve the ID of this tag to see which records it is applied to. You can execute the following in the AJAX Toolkit or your own client application:

To update the tag definition:

1. Query for all the tags with the name WC.

```
var IDToUpdate = sforce.connection.query("SELECT Id FROM TagDefinition " +  
    "WHERE Name = 'WC'");
```

2. The name WC should be West Coast. Since this change affects multiple record types, create a new TagDefinition record with the correct value.

```
var updateTD = new sforce.SObject('TagDefinition');  
updateTD.Id = IDToUpdate.records.Id;  
IDToUpdate.records.Name = "West Coast";
```

3. updateTD is used to replace the previous TagDefinition record. The final step is to call update() using the corrected record.

```
sforce.connection.update([updateTD]);
```

Since we essentially overwrote the ID, every record previously tagged with WC will now be tagged as West Coast.

## Discussion

Operating on TagDefinition records does not take into consideration the types of any child tags. Ensure that the action you want to perform is appropriate for all types of tags. For instance, an AccountTag named WC and an ContactTag named WC will both be changed to West Coast. For information on changing tags on multiple records, see [Add and Remove Tags on Multiple Records](#) on page 263.

## See Also

- [Viewing Tags](#) on page 37
- [Viewing Records with Tags](#) on page 38
- “TagDefinition” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/Content/sforce\\_api\\_objects\\_tagdefinition.htm](http://www.salesforce.com/us/developer/docs/api/Content/sforce_api_objects_tagdefinition.htm)
- “update ()” in the *Force.com Web Services API Developer’s Guide* at [www.salesforce.com/us/developer/docs/api/index\\_CSH.htm#sforce\\_api\\_calls\\_update.htm](http://www.salesforce.com/us/developer/docs/api/index_CSH.htm#sforce_api_calls_update.htm)

# Force.com Platform Glossary

---

[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](#)

This glossary defines terms that appear throughout the Force.com platform documentation suite.

## A

### **Activity (Calendar Events/Tasks)**

Planned task or event, optionally related to another type of record such as an account, contact, lead, opportunity, or case.

### **Activity History**

The Activity History related list of a record displays all completed tasks, logged phone calls, expired events, outbound email, mass email, email added from Microsoft Outlook®, and merged documents for the record and its associated records.

### **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.

### **Advanced Function**

A formula function designed for use in custom buttons, links, and s-controls. For example, the `INCLUDE` advanced function returns the content from an s-control snippet.

### **AJAX Toolkit**

A JavaScript wrapper around the API that allows you to execute any API call and access any object you have permission to view from within JavaScript code. For more information, see the [\*AJAX Toolkit Developer's Guide\*](#).

### **Analytic Snapshot**

An analytic snapshot enables users to run a tabular or summary report and save the report results to fields on a custom object. With analytic snapshots, users with the appropriate permissions can map fields from a source report to the fields on a target object, and schedule when to run the report to load the custom object's fields with the report's data.

**Analytic Snapshot Running User**

The user whose security settings determine the source report's level of access to data. This bypasses all security settings, giving all users who can view the results of the source report in the target object access to data they might not be able to see otherwise.

**Analytic Snapshot Source Report**

The custom report scheduled to run and load data as records into a custom object.

**Analytic Snapshot Target Object**

The custom object that receives the results of the source report as records.

**Anonymous Block, Apex**

An Apex script that does not get stored in Salesforce.com, but that can be compiled and executed through the use of the `ExecuteAnonymousResult()` API call, or the equivalent in the [AJAX Toolkit](#).

**Anti-Join**

An anti-join is a subquery on another object in a `NOT IN` clause in a SOQL query. You can use anti-joins to create advanced queries, such as getting all accounts that do not have any open opportunities. See also Semi-Join.

**Apex**

Force.com Apex code 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 code enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex scripts can be initiated by Web service requests and from triggers on objects.

**Apex Controller**

See Controller, Visualforce.

**Apex-Managed Sharing**

Enables developers to programmatically manipulate sharing to support their application's behavior. Apex-managed sharing is only available for custom objects.

**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.com 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 AppExchange along with related components such as custom fields, custom tabs, and custom objects. Then, you can make the app available to other Salesforce.com users from AppExchange.

**App Menu**

See Force.com App Menu.

**AppExchange**

AppExchange is a sharing interface from salesforce.com that allows you to browse and share apps and services for the Force.com platform.

**AppExchange Upgrades**

Upgrading an app is the process of installing a newer version.

**Application Lifecycle Management (ALM)**

The process of managing an application's lifecycle, from planning, to development, to integration and support. Application lifecycle management is often depicted as a circle, because the cycle repeats itself as the application evolves.

**Application Programming Interface (API)**

The interface that a computer system, library, or application provides in order to allow other computer programs to request services from it and exchange data between them.

**Approval Action**

See Workflow and Approval Actions.

**Approval Process**

An automated process your organization can use to approve records in Salesforce.com. An approval process specifies the steps necessary for a record to be approved and who must approve it at each step. A step can apply to all records included in the process, or just records that have certain attributes. An approval process also specifies the actions to take when a record is approved, rejected, recalled, or first submitted for approval.

**Asynchronous Calls**

A call that does not return results immediately because the operation may take a long time. Calls in the Metadata API and Bulk API are asynchronous.

## **Auto Number**

A custom field type that automatically adds a unique sequential number to each record. These fields are read only.

## **B**

### **Batch Apex**

The ability to perform long, complex operations on many records at a scheduled time using Apex.

### **Batch, Bulk API**

A batch is a CSV or XML representation of a set of records in the Bulk API. You process a set of records by creating a job that contains one or more batches. Each batch is processed independently by the server, not necessarily in the order it is received. See Job, Bulk API.

### **Beta, Managed Package**

In the context of managed packages, a beta managed package is an early version of a managed package distributed to a sampling of your intended audience to test it.

### **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.

### **Bulk API**

The REST-based Bulk API is optimized for processing large sets of data. It allows you to insert, update, upsert, or delete a large number of records asynchronously by submitting a number of batches which are processed in the background by Salesforce.com.

## **C**

### **Cascading Style Sheet (CSS)**

Files that contain all of the information relevant to color, font, borders, and images that are displayed in a user interface.

### **Child Relationship**

A relationship that has been defined on an sObject that references another sObject as the “one” side of a one-to-many relationship. For example, contacts, opportunities, and tasks have child relationships with accounts. See also sObject.

### **Class, Apex**

A template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types,

and static initialization code. In most cases, Apex classes are modeled on their counterparts in Java.

### **Client App**

An app that runs outside the Salesforce.com user interface and uses only the Force.com API or Bulk API. It typically runs on a desktop or mobile device. These apps treat the platform as a data source, using the development model of whatever tool and platform for which they are designed. See also Composite App and Native App.

### **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.

### **Cloud Computing**

A virtual development and deployment environment that eliminates the need for computing hardware and software infrastructure and components. Developers and users connect to this environment through a Web browser.

### **Code Coverage**

A way to identify which lines of code are exercised by a set of unit tests, and which are not. This helps you identify sections of code that are completely untested and therefore at greatest risk of containing a bug or introducing a regression in the future.

### **Collapsible Section**

Sections on detail pages that users can hide or show.

### **Combination Chart**

A combination chart plots multiple sets of data on a single chart. Each set of data is based on a different field, so values are easy to compare. You can also combine certain chart types to present data in different ways on a single chart.

### **Component, Metadata**

A component is an instance of a metadata type in the Metadata API. For example, CustomObject is a metadata type for custom objects, and the MyCustomObject\_\_c component is an instance of a custom object. A component is described in an XML file and it can be deployed or retrieved using the Metadata API, or tools built on top of it, such as the Force.com IDE or the Force.com Migration Tool.

### **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*.

**Composite App**

An app that combines native platform functionality with one or more external Web services, such as Yahoo! Maps. Composite apps allow for more flexibility and integration with other services, but may require running and managing external code. See also Client App and Native App.

**Connect for Lotus Notes**

Force.com Connect for Lotus Notes is an add-in for IBM® Lotus Notes® that adds new commands to the Lotus Notes user interface that let you interact with Salesforce.com.

**Connect for Office**

Product that allows you to integrate Salesforce.com with Microsoft® Word and Excel.

**Connect for Outlook**

Force.com Connect for Microsoft Outlook is an add-in for Microsoft Outlook that allows you to interact with Salesforce.com conveniently from Outlook. Connect for Outlook adds buttons and options to your Outlook user interface.

**Connect Offline**

Product that allows salespeople to use Salesforce.com to update their data remotely, anywhere, anytime—totally unplugged.

**Console**

A page that combines related records into one screen with different frames so that users can view and edit information all in one place.

**Console Layout**

Objects chosen by an administrator to display in the list view frame of the console. For example, if an administrator adds cases to a console layout, then users whose profiles are assigned to that console layout can see list views of cases in the console's list view frame.

**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.

**Controlling Field**

Any standard or custom picklist or checkbox field whose values control the available values in one or more corresponding dependent fields.

**Cookie**

Client-specific data used by some Web applications to store user and session-specific information. Salesforce.com issues a session “cookie” only to record encrypted authentication information for the duration of a specific session.

**CSV (Comma Separated Values)**

A file format that enables the sharing and transportation of structured data. The import wizards, Data Loader and the Bulk API support CSV. Each line in a CSV file represents a record. A comma separates each field value in the record.

**Custom App**

See App.

**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 profile-based 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.com 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 URLs defined by administrators to integrate your Salesforce.com 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 Report Type**

A selection in the Report Wizard that allows you to define the report criteria from which users can run and create custom reports. For example,

you may want users to run and customize reports based on accounts with or without opportunities. With custom report types, you can enable users to create reports from predefined standard and custom objects, object relationships, and standard and custom fields that you specify..

### **Custom Settings**

Custom settings are similar to custom objects and enable application developers to create custom sets of data, as well as create and associate custom data for an organization, profile, or specific user. All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. This data can then be used by formula fields, validation rules, Apex, and the Force.com Web Services API.

See also [Hierarchy Custom Settings](#) and [List Custom Settings](#).

### **Custom S-Control**

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 View**

A display feature that lets you see a specific set of records for a particular object.

## **D**

### **Dashboard**

A visual representation of your custom report data. You can create several dashboard components within a dashboard to give you a real-time snapshot of corporate metrics and key performance indicators.

### **Database**

An organized collection of information. The underlying architecture of the Force.com platform includes a database where your data is stored.

### **Database Table**

A list of information, presented with rows and columns, about the person, thing, or concept you want to track. See also [Object](#).

### **Data Loader**

A Force.com platform tool used to import and export data from your Salesforce.com organization.

### **Data Manipulation Language (DML)**

An Apex method or operation that inserts, updates, or deletes records from the Force.com platform database.

**Data State**

The structure of data in an object at a particular point in time.

**Date Literal**

A keyword in a SOQL or SOSL query that represents a relative range of time such as `last month` or `next year`.

**Decimal Places**

Parameter for number, currency, and percent custom fields that indicates the total number of digits you can enter to the right of a decimal point, for example, 4.98 for an entry of 2. Note that the system rounds the decimal numbers you enter, if necessary. For example, if you enter 4.986 in a field with `Decimal Places` of 2, the number rounds to 4.99.

**Delegated Administration**

A security model in which a group of non-administrator users perform administrative tasks.

**Delegated Authentication**

A security process where an external authority is used to authenticate Force.com platform users.

**Dependency**

A relationship where one object's existence depends on that of another. There are a number of different kinds of dependencies including mandatory fields, dependent objects (parent-child), file inclusion (referenced images, for example), and ordering dependencies (when one object must be deployed before another object).

**Dependent Field**

Any custom picklist or multi-select picklist field that displays available values based on the value selected in its corresponding controlling field.

**Deploy**

To move functionality from an inactive state to active. For example, when developing new features in the Salesforce.com user interface, you must select the “Deployed” option to make the functionality visible to other users.

The process by which an application or other functionality is moved from development to production.

To move metadata components from a local file system to a Salesforce.com organization.

For installed apps, deployment makes any custom objects in the app available to users in your organization. Before a custom object is deployed, it is only available to administrators and any users with the “Customize Application” permission.

**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.

**Developer Edition**

A free, fully-functional Salesforce.com organization designed for developers to extend, integrate, and develop with the Force.com platform. Developer Edition accounts are available on [developer.force.com](http://developer.force.com).

**Developer Force**

The Developer Force website at [developer.force.com](http://developer.force.com) provides a full range of resources for platform developers, including sample code, toolkits, an online developer community, and the ability to obtain limited Force.com platform environments.

**Development as a Service (DaaS)**

An application development model where all development is on the Web. This means that source code, compilation, and development environments are not on local machines, but are Web-based services.

**Development Environment**

A Salesforce.com organization where you can make configuration changes that will not affect users on the production organization. There are two kinds of development environments, sandboxes and Developer Edition organizations.

**Document Library**

A place to store documents without attaching them to accounts, contacts, opportunities, or other records.

**E****Email Alert**

Email alerts are workflow and approval actions that are generated using an email template by a workflow rule or approval process and sent to designated recipients, either Salesforce.com users or others.

**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.

**Enterprise Application**

An application that is designed to support functionality for an organization as a whole, rather than solving a specific problem.

**Enterprise Edition**

A Salesforce.com edition designed for larger, more complex businesses.

**Enterprise WSDL**

A strongly-typed WSDL for customers who want to build an integration with their Salesforce.com organization only, or for partners who are using tools like Tibco or webMethods to build integrations that require strong typecasting. The downside of the Enterprise WSDL is that it only works with the schema of a single Salesforce.com organization because it is bound to all of the unique objects and fields that exist in that organization's data model.

**Entity Relationship Diagram (ERD)**

A data modeling tool that helps you organize your data into entities (or objects, as they are called in the Force.com platform) and define the relationships between them. ERD diagrams for key Salesforce.com objects are published in the *Force.com Web Services API Developer's Guide*.

**Enumeration Field**

An enumeration is the WSDL equivalent of a picklist field. The valid values of the field are restricted to a strict set of possible values, all having the same data type.

**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**

A part of an object that holds a specific piece of information, such as a text or currency value.

**Field-Level Security**

Settings that determine whether fields are hidden, visible, read only, or editable for users based on their profiles. Available in Enterprise, Unlimited, and Developer Editions only.

**Field Dependency**

A filter that allows you to change the contents of a picklist based on the value of another field.

**Field Update**

Field updates are workflow and approval actions that specify the field you want updated and the new value for it. Depending on the type of field, you can choose to apply a specific value, make the value blank, or calculate a value based on a formula you create.

**Filter Condition/Criteria**

Condition on particular fields that qualifies items to be included in a list view or report, such as “State equals California.”

**Flex Toolkit for Force.com**

An Adobe® Flex™ library that allows you to access Salesforce.com data from within a Flex 2 application.

**Folder**

A central place to store documents, email templates, and reports. Folders are available for all organizations. Report and document folders are not available in Personal Edition.

**Force.com**

The salesforce.com platform for building applications in the cloud. Force.com combines a powerful user interface, operating system, and database to allow you to customize and deploy applications in the cloud for your entire enterprise.

**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.

**Force.com Builder**

The App Setup area of the Salesforce.com setup menu.

**Force.com IDE**

An Eclipse plug-in that allows developers to manage, author, debug and deploy Force.com applications in the Eclipse development environment.

**Force.com Migration Tool**

A toolkit that allows you to write an Apache Ant build script for migrating Force.com components between a local file system and a Salesforce.com organization.

**Force.com Web Services API**

A Web services-based application programming interface that provides access to your Salesforce.com organization's information.

**Foreign key**

A field whose value is the same as the primary key of another table. You can think of a foreign key as a copy of a primary key from another table.

A relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

### **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.

### **Global Variable**

A special merge field that you can use to reference data in your organization.

A method access modifier for any method that needs to be referenced outside of the application, either in the Force.com Web Services API or by other Apex scripts.

### **Governor limits**

Apex execution limits that prevent developers who write inefficient code from monopolizing the resources of other Salesforce.com users.

### **Gregorian Year**

A calendar based on a twelve month structure used throughout much of the world.

### **Group**

A group is a set of users. Groups can contain individual users, other groups, or the users in a role. Groups can be used to help define sharing access to data or to specify which data to synchronize when using Connect for Outlook or Connect for Lotus Notes.

Users can define their own personal groups. Administrators can create public groups for use by everyone in the organization.

### **Group Edition**

A product designed for small businesses and workgroups with a limited number of users.

**H****Hierarchy Custom Settings**

A type of custom setting that uses a built-in hierarchical logic that lets you “personalize” settings for specific profiles or users. The hierarchy logic checks the organization, profile, and user settings for the current user and returns the most specific, or “lowest,” value. In the hierarchy, settings for an organization are overridden by profile settings, which, in turn, are overridden by user settings.

**Home Organization**

The organization used for retrieving components to a local file system. If using the Force.com IDE, the home organization is the organization used to create a project. If using the Force.com Migration Tool, the home organization is the server you specify in build.properties when you retrieve components.

**Home Tab**

Starting page from which users can choose sidebar shortcuts and options, view current tasks and activities, or select another tab.

**Hover Detail**

Hover details display an interactive overlay containing detailed information about a record when users hover the mouse over a link to that record in the Recent Items list on the sidebar or in a lookup field on a record detail page. Users can quickly view information about a record before clicking **View** for the record's detail page or **Edit** for the edit page. The fields displayed in the hover details are determined by the record's mini page layout. The fields that display in document hover details are not customizable.

**HTML S-Control**

An s-control that contains the actual HTML that should be rendered on a page. When saved this way, the HTML is ultimately hosted on a platform server, but is executed in an end-user's browser.

**HTTP Debugger**

An application that can be used to identify and inspect SOAP requests that are sent from the AJAX Toolkit. They behave as proxy servers running on your local machine and allow you to inspect and author individual requests.

**I****ID**

See Salesforce.com Record ID.

**IdeaExchange**

A forum where salesforce.com customers can suggest new product concepts, promote favorite enhancements, interact with product managers and other customers, and preview what salesforce.com is planning to deliver in future releases. Visit IdeaExchange at [ideas.salesforce.com](https://ideas.salesforce.com).

**Integrated Development Environment (IDE)**

A software application that provides comprehensive facilities for software developers including a source code editor, testing and debugging tools, and integration with source code control systems.

**Immediate Action**

A workflow action that executes instantly when the conditions of a workflow rule are met.

**Import Wizard**

A tool for importing data into your Salesforce.com organization, accessible from Setup.

**Inline S-Control**

An s-control that displays within a record detail page or dashboard, rather than on its own page.

**Instance**

The cluster of software and hardware represented as a single logical server that hosts an organization's data and runs their applications. The Force.com platform runs on multiple instances, but data for any single organization is always consolidated on a single instance.

**Integration Testing**

An intermediate phase of software testing in which individual projects are combined and tested as a group. Integration testing follows unit testing and precedes system testing.

**Integration User**

A Salesforce.com user defined solely for client apps or integrations. Also referred to as the logged-in user in a Force.com Web Services API context.

**ISO Code**

The International Organization for Standardization country code, which represents each country by two letters.

**J****Job, Bulk API**

A job in the Bulk API specifies which object is being processed (for example, Account, Opportunity) and what type of action is being used (insert, upsert, update, or delete). You process a set of records by creating a job that contains one or more batches. See Batch, Bulk API.

## **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**

### **Layout**

See Page Layout.

### **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.

### **Letterhead**

Determines the basic attributes of an HTML email template. Users can create a letterhead that includes attributes like background color, logo, font size, and font color.

### **License Management Application (LMA)**

A free AppExchange app that allows you to track sales leads and accounts for every user who downloads a managed package of yours from AppExchange.

### **License Management Organization (LMO)**

The Salesforce.com organization that you use to track all the Salesforce.com users who install your package. A license management organization must have the License Management Application (LMA) installed. It automatically receives notification every time your package is installed or uninstalled so that you can easily notify users of upgrades. You can specify any Enterprise, Unlimited, or Developer Edition organization as your license management organization. For more information, go to <http://www.salesforce.com/docs/en/lma/index.htm>.

### **List Custom Settings**

A type of custom setting that provides a reusable set of static data that can be accessed across your organization. If you use a particular set of data frequently within your application, putting that data in a list custom setting streamlines access to it. Data in list settings does not vary with profile or

user, but is available organization-wide. Examples of list data include two-letter state abbreviations, international dialing prefixes, and catalog numbers for products. Because the data is cached, access is low-cost and efficient: you don't have to use SOQL queries that count against your governor limits.

### List View

A list display of items (for example, accounts or contacts) based on specific criteria. Salesforce.com provides some predefined views.

In the console, the list view is the top frame that displays a list view of records based on specific criteria. The list views you can select to display in the console are the same list views defined on the tabs of other objects. You cannot create a list view within the console.

### Local Name

The value stored for the field in the user's or account's language. The local name for a field is associated with the standard name for that field.

### Local Project

A .zip file containing a project manifest (package.xml file) and one or more metadata components.

### Locale

The country or geographic region in which the user is located. The setting affects the format of date and number fields, for example, dates in the English (United States) locale display as 06/30/2000 and as 30/06/2000 in the English (United Kingdom) locale.

In Professional, Enterprise, Unlimited, and Developer Edition organizations, a user's individual Locale setting overrides the organization's Default Locale setting. In Personal and Group Editions, the organization-level locale field is called Locale, not Default Locale.

### Long Text Area

Data type of custom field that allows entry of up to 32,000 characters on separate lines.

### Lookup Dialog

Popup dialog available for some fields that allows you to search for a new item, such as a contact, account, or user.

### Lookup Field

A type of field that contains a linkable value to another record. You can display lookup fields on page layouts where the object has a lookup or master-detail relationship with another object. For example, cases have a lookup relationship with assets that allows users to select an asset using a lookup dialog from the case edit page and click the name of the asset from the case detail page.

**Lookup Relationship**

A relationship between two records so you can associate records with each other. For example, cases have a lookup relationship with assets that lets you associate a particular asset with a case. On one side of the relationship, a lookup field allows users to click a lookup icon and select another record from a popup window. On the associated record, you can then display a related list to show all of the records that have been linked to it. A lookup relationship has no effect on record deletion or security, and the lookup field is not required in the page layout.

**M****Managed Package**

A collection of application components that are posted as a unit on AppExchange, and are associated with a namespace and possibly a License Management Organization. A package must be managed for it to support upgrades. An organization can create a single managed package that can be downloaded and installed by many different organizations. They differ from unmanaged packages in that some components are locked, allowing the managed package to be upgraded later. Unmanaged packages do not include locked components and cannot be upgraded. In addition, managed packages obfuscate certain components (like Apex) on subscribing organizations, so as to protect the intellectual property of the developer.

**Managed Package Extension**

Any package, component, or set of components that adds to the functionality of a managed package. An extension requires that the base managed package be installed in the organization.

**Manifest File**

The project manifest file (`package.xml`) lists the XML components to retrieve or deploy when working with the Metadata API, or clients built on top of the Metadata API, such as the Force.com IDE or the Force.com Migration Tool.

**Manual Sharing**

Record-level access rules that allow record owners to give read and edit permissions to other users who might not have access to the record any other way.

**Many-to-Many Relationship**

A relationship where each side of the relationship can have many children on the other side. Many-to-many relationships are implemented through the use of junction objects.

**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.

### **Master Picklist**

A complete list of picklist values available for a record type or business process.

### **Matrix Report**

Matrix reports are similar to summary reports, but allow you to group and summarize data by both rows and columns. They can be used as the source report for dashboard components. Use this type for comparing related totals, especially if you have large amounts of data to summarize and you need to compare values in several different fields, or you want to look at data by date *and* by product, person, or geography.

### **Merge Field**

A field you can place 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 First Name field to address an email recipient by his or her first name.

### **Metadata**

Information about the structure, appearance, and functionality of an organization and any of its parts. Force.com uses XML to describe metadata.

### **Metadata Component**

An instance of a metadata type. For example, the Account standard object is a metadata component, which is an instance of the CustomObject type.

### **Metadata-Driven Development**

An app development model that allows apps to be defined as declarative “blueprints,” with no code required. Apps built on the platform—their data models, objects, forms, workflows, and more—are defined by metadata.

### **Metadata Type**

A type of object in a Salesforce.com organization, for example CustomObject or CustomTab.

### **Metadata WSDL**

A WSDL for users who want to use the Force.com Metadata API calls.

### **Migration**

The process of moving metadata from one organization to another, usually between two development environments. See also Deploy.

**Mini Page Layout**

A subset of the items in a record's existing page layout that administrators choose to display in the console's Mini View and in Hover Details. Mini page layouts inherit record type and profile associations, related lists, fields, and field access settings from the page layout.

**Mini View**

The console's right frame which displays the records associated with the record displayed in the detail view. The fields displayed in the mini view are defined in the mini page layouts by an administrator. The mini view does not display if the record in the detail view does not have any records associated with it.

**Multitenancy**

An application model where all users and apps share a single, common infrastructure and code base.

**MVC (Model-View-Controller)**

A design paradigm that deconstructs applications into components that represent data (the model), ways of displaying that data in a user interface (the view), and ways of manipulating that data with business logic (the controller).

**N****Namespace**

In a packaging context, a one- to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers on AppExchange, similar to a domain name. Salesforce.com automatically prepends your namespace prefix, followed by two underscores (“\_\_”), to all unique component names in your Salesforce.com organization.

**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.com organizations. It keeps your managed package under your control exclusively.

**Native App**

An app that is built exclusively with setup (metadata) configuration on Force.com. Native apps do not require any external services or infrastructure.

**Notes**

Miscellaneous information pertaining to a specific record.

**O****Object**

An object allows you to store information in your Salesforce.com 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.

**Object-Level Security**

Settings that allow an administrator to hide whole tabs and objects from a user so that he or she does not know that type of data exists. On the platform you set object-level access rules with object permissions on user profiles.

**onClick JavaScript**

JavaScript code that executes when a button or link is clicked.

**One-to-Many Relationship**

A relationship in which a single object is related to many other objects. For example, an account may have one or more related contacts.

**Organization**

A deployment of Salesforce.com with a defined set of licensed users. An organization is the virtual space provided to an individual customer of salesforce.com. Your organization includes all of your data and applications, and is separate from all other organizations.

**Organization-Wide Defaults**

Settings that allow you to specify the baseline level of data access that a user has in your organization. For example, you can make it so that any user can see any record of a particular object that is enabled in their user profile, but that they need extra permissions to edit one.

**Outbound Call**

Any call that originates from a user to a number outside of a call center in Salesforce CRM Call Center.

**Outbound Message**

Workflow, approval, or milestone actions that send 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.com setup menu. Then you must configure the external endpoint. You can create a listener for the messages using the Force.com Web Services API.

### **Overlay**

An overlay displays additional information when you hover your mouse over certain user interface elements. Depending on the overlay, it will close when you move your mouse away, click outside of the overlay, or click a close button.

### **Owner**

Individual user to which a record (for example, a contact or case) is assigned.

## **P**

### **PaaS**

See Platform as a Service.

### **Package**

A group of Force.com components and applications that are made available to other organizations through the AppExchange. You use packages to bundle an app along with any related components so that you can upload them to AppExchange together.

### **Package Dependency**

Created when one component references another component, permission, or preference, which must exist for the component to be valid. Components can include, but are not limited to:

- Standard or custom fields
- Standard or custom objects
- Visualforce pages
- Apex scripts

Permissions and preferences can include but are not limited to:

- Divisions
- Multicurrency
- Record types

### **Package Installation**

Incorporates the contents of a package into your Salesforce.com organization. A package on AppExchange can include an app, a component, or a combination of the two. After you install a package, you may need to deploy components in the package to make it generally available to the users in your organization.

**Package Publication**

Publishing your package makes it publicly available on AppExchange. Apps can be found under specific categories and by doing a search for keywords.

**Page Layout**

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.

**Parameterized Typing**

Parameterized typing allows interfaces to be implemented with generic data type parameters that are replaced with actual data types upon construction.

**Parent Account**

Organization or company that an account is affiliated with or owned by. By specifying a parent for an account, you can get a global view of all parent/subsidiary relationships using the **View Hierarchy** link.

**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.

**Partner WSDL**

A loosely-typed WSDL for customers, partners, and ISVs who want to build an integration or an AppExchange app that can work across multiple Salesforce.com organizations. With this WSDL, the developer is responsible for marshaling data in the correct object representation, which typically involves editing the XML. However, the developer is also freed from being dependent on any particular data model or Salesforce.com organization. Contrast this with the Enterprise WSDL, which is strongly typed.

**Personal Edition**

Product designed for individual sales representatives and single users.

**Personal Information**

User information including personal contact information, quotas, personal group information, and default sales team.

**Picklist**

Selection list of options available for specific fields in a Salesforce.com object, for example, the `Industry` field for accounts. Users can choose a single value from a list of options rather than make an entry directly in the field. See also Master Picklist.

**Picklist (Multi-Select)**

Selection list of options available for specific fields in a Salesforce.com object. Multi-select picklists allow users to choose one or more values. Users can choose a value by double clicking on it, or choose additional values from a scrolling list by holding down the Control key while clicking a value and using the arrow icon to move them to the selected box.

**Picklist Values**

Selections displayed in drop-down lists for particular fields. Some values come predefined, and other values can be changed or defined by an administrator.

**Platform as a Service (PaaS)**

An environment where developers use programming tools offered by a service provider to create applications and deploy them in a cloud. The application is hosted as a service and provided to customers via the Internet. The PaaS vendor provides an API for creating and extending specialized applications. The PaaS vendor also takes responsibility for the daily maintenance, operation, and support of the deployed application and each customer's data. The service alleviates the need for programmers to install, configure, and maintain the applications on their own hardware, software, and related IT resources. Services can be delivered using the PaaS environment to any market segment.

**Platform Edition**

A Salesforce.com edition based on either Enterprise Edition or Unlimited Edition that does not include any of the standard Salesforce.com CRM apps, such as Sales or Service & Support.

**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 Key**

A relational database concept. Each table in a relational database has a field in which the data value uniquely identifies the record. This field is called the primary key. The relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

**Printable View**

An option that displays a page in a print-ready format.

**Private Sharing**

Private sharing is the process of sharing an uploaded package by using the URL you receive from Salesforce.com. This URL is not listed in the AppExchange. Using the unlisted URL allows you to share a package without going through the listing process or making it public.

**Process Visualizer**

A tool that displays a graphical version of an approval process. The view-only diagram is presented as a flowchart. The diagram and an informational sidebar panel can help you visualize and understand the defined steps, rule criteria, and actions that comprise your approval process.

**Production Organization**

A Salesforce.com organization that has live users accessing data.

**Professional Edition**

A Salesforce.com edition designed for businesses who need full-featured CRM functionality.

**Profile**

Defines a user's permission to perform different functions within Salesforce.com. For example, the Solution Manager profile gives a user access to create, edit, and delete solutions.

**Project**

See Local Project.

**Project Manifest**

A control file (`package.xml`) that determines which components are retrieved or deployed. See also Local Project.

**Prototype**

The classes, methods and variables that are available to other Apex scripts.

**Provider**

The provider of an AppExchange listing is the Salesforce.com user or organization that published the listing.

**Public Calendar**

A calendar in which a group of people can track events of interest to all of them (such as marketing events, product releases, or training classes) or schedule a common activity (such as a team vacation calendar). For example, your marketing team can set up an events calendar to show upcoming marketing events to the entire sales and marketing organization.

**Q****Queue**

A holding area for items before they are processed. Salesforce.com uses queues in a number of different features and technologies.

**Query Locator**

A parameter returned from the `query()` or `queryMore()` API call that specifies the index of the last result record that was returned.

**Query String Parameter**

A name-value pair that's included in a URL, typically after a '?' character. For example:

```
http://na1.salesforce.com/001/e?name=value
```

**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.)

**Recent Items**

List of links in the sidebar for most recently accessed records. Note that not all types of records are listed in the recent items.

**Record**

A single instance of a Salesforce.com object. For example, "John Jones" might be the name of a contact record.

**Record ID**

See Salesforce.com Record ID.

**Record-Level Security**

A method of controlling data in which you can allow a particular user to view and edit an object, but then restrict the records that the user is allowed to see.

**Record Name**

A standard field on all Salesforce.com objects. Whenever a record name is displayed in a Force.com application, the value is represented as a link to a detail view of the record. A record name can be either free-form text or an autonumber field. Record Name does not have to be a unique value.

**Record Type**

A field available for certain records that can include some or all of the standard and custom picklist values for that record. Record types are special fields that you can associate with profiles to make only the included picklist values available to users with that profile.

**Recycle Bin**

A page that lets you view and restore deleted information. Access the Recycle Bin by using the link in the sidebar.

**Regression Testing**

Testing that uncovers why functionality that was previously working stops working as intended. A stable set of data must be used for regression testing. For this reason, it is a good idea to use a configuration-only sandbox that pulls its data from an immutable source. See also Sandbox Organization.

**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 List Hover Links**

A type of link that allows you to quickly view information on a detail page about related lists, by hovering your mouse over the link. Your administrator must enable the display of hover links. The displayed text contains the corresponding related list and its number of records. You can also click this type of link to jump to the content of the related list without having to scroll down the page.

**Related Object**

Objects chosen by an administrator to display in the console'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.

**Relationship Query**

In a SOQL context, a query that traverses the relationships between objects to identify and return results. Parent-to-child and child-to-parent syntax differs in SOQL queries.

**Release Management**

See Application Lifecycle Management (ALM).

**Release Train**

A scheduling technique for delivering application upgrades on a regular cycle.

**Remote Access Application**

A remote access application is an application external to Salesforce.com that uses the OAuth protocol to verify both the Salesforce.com user and the external application.

## Report

Information you can display or print that provides a summary of your data.  
See Tabular Report, Summary Report, and Matrix Report.

## Report Types

Specifies the objects and fields that can be used as the basis of a report.  
In addition to pre-defined standard report types, you can create custom report types for more advanced reporting requirements.

## Role Hierarchy

A record-level security setting that defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.

## Roll-Up Summary Field

A field type that automatically provides aggregate values from child records in a master-detail relationship.

## Running User

The user whose security settings determine what data is displayed in a dashboard. Because only one running user is specified per dashboard, everyone who can access the dashboard sees the same data, regardless of their personal security settings.

# S

## SaaS

See Software as a Service (SaaS).

## S-Control

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.



**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.

## Salesforce.com API Version

See Version.

## Salesforce CRM Call Center

A Salesforce.com feature that seamlessly integrates Salesforce.com with third-party computer-telephony integration (CTI) systems.

## Salesforce CRM Content

An on-demand, content-management system that allows you to organize, share, search, and manage content within your organization and across key areas of the Salesforce.com application. Content can include all file types, from traditional business documents such as Microsoft PowerPoint presentations to audio files, video files, and Web pages.

## Salesforce CRM and Google AdWords

Salesforce CRM and Google AdWords is a Force.com AppExchange app that connects Google AdWords with Salesforce.com, allowing you to track the effectiveness of your online advertising investments.

## Salesforce CRM for Outlook

Salesforce CRM for Outlook is a Microsoft® Outlook® integration application that lets you log emails in Salesforce.com and sync calendar events and contacts between Outlook and Salesforce.com. It consists of a system tray program and an Outlook add-in. User settings are stored in Outlook configurations, which administrators manage in Salesforce.com.



**Note:** Salesforce CRM for Outlook is available through a pilot program. For information on enabling Salesforce CRM for Outlook for your organization, contact [salesforce.com](http://salesforce.com).

## Salesforce Mobile

Salesforce Mobile is a Salesforce.com feature that enables users to access their Salesforce.com data from mobile devices running the mobile client application. The Salesforce Mobile client application exchanges data with Salesforce.com over wireless carrier networks, and stores a local copy of the user's data in its own database on the mobile device. Users can edit local copies of their Salesforce.com records when a wireless connection is unavailable, and transmit those changes when a wireless connection becomes available..

## Salesforce.com Record ID

A unique 15- or 18-character alphanumeric string that identifies a single record in Salesforce.com.

## Salesforce.com SOA (Service-Oriented Architecture)

A powerful capability of Force.com that allows you to make calls to external Web services from within Apex.

## Sandbox Organization

A nearly identical copy of a Salesforce.com production organization. You can create multiple sandboxes in separate environments for a variety of purposes, such as testing and training, without compromising the data and applications in your production environment.

### **Save As**

Option on any standard, public, or custom report to save the parameters of the report without altering the original report. It creates a new custom report with your saved changes.

### **Save & New**

Alternative “save” on most pages with which you can save your current changes and create a new entry.

### **Search**

Feature that lets you search for information that matches specified keywords. You can enter a search from the Search section of the sidebar column, or perform an Advanced Search by clicking **Advanced Search** in the sidebar.

### **Search Layout**

The organization of fields included in search results, in lookup dialogs, and in the key lists on tab home pages.

### **Search Phrase**

Search phrases are queries that users enter when searching on [www.google.com](http://www.google.com).

### **Service**

A service is an offering of professional assistance. Services related to Salesforce.com and the Force.com platform, such as enhanced customer support, or assistance with configuration can be listed on platform. can be listed on AppExchange.

### **Semi-Join**

A semi-join is a subquery on another object in an `IN` clause in a SOQL query. You can use semi-joins to create advanced queries, such as getting all contacts for accounts that have an opportunity with a particular record type. See also Anti-Join.

### **Session ID**

An authentication token that is returned when a user successfully logs in to Salesforce.com. The Session ID prevents a user from having to log in again every time he or she wants to perform another action in Salesforce.com. Different from a record ID or Salesforce.com ID, which are terms for the unique ID of a Salesforce.com record.

### **Session Timeout**

The period of time after login before a user is automatically logged out. Sessions expire automatically after a predetermined length of inactivity, which can be configured in Salesforce.com by clicking **Setup > Security Controls**. The default is 120 minutes (two hours). The inactivity timer is reset to zero if a user takes an action in the Web interface or makes an API call.

**Setter Methods**

Methods that assign values. See also Getter Methods.

**Setup**

An administration area where you can customize and define Force.com applications. Access Setup through the **Setup** link at the top of Salesforce.com pages.

**Sharing**

Allowing other users to view or edit information you own. There are different ways to share data:

- Sharing Model—defines the default organization-wide access levels that users have to each other's information and whether to use the hierarchies when determining access to data.
- Role Hierarchy—defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.
- Sharing Rules—allow an administrator to specify that all information created by users within a given group or role is automatically shared to the members of another group or role.
- Manual Sharing—allows individual users to share a specific account or opportunity with other users or groups.
- Apex-Managed Sharing—enables developers to programmatically manipulate sharing to support their application's behavior. See Apex-Managed Sharing.

**Sharing Model**

Behavior defined by your administrator that determines default access by users to different types of records.

**Sharing Rule**

Type of default sharing created by administrators. Allows users in a specified group or role to have access to all information created by users within a given group or role.

**Show/Hide Details**

Option available for reports that lets you show/hide the details of individual column values in report results.

**Sidebar**

Column appearing on the left side of each page that provides links to search, recent items, and other resources.

## Sites

Force.com sites enables you to create public websites and applications that are directly integrated with your Salesforce.com 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.

## Snippet

A type of s-control that is designed to be included in other s-controls. Similar to a helper method that is used by other methods in a piece of code, a snippet allows you to maintain a single copy of HTML or JavaScript that you can reuse in multiple s-controls.

## SOAP (Simple Object Access Protocol)

A protocol that defines a uniform way of passing XML-encoded data.

## sObject

Any object that can be stored in the Force.com platform.

## Software as a Service (SaaS)

A delivery model where a software application is hosted as a service and provided to customers via the Internet. The SaaS vendor takes responsibility for the daily maintenance, operation, and support of the application and each customer's data. The service alleviates the need for customers to install, configure, and maintain applications with their own hardware, software, and related IT resources. Services can be delivered using the SaaS model to any market segment.

## SOQL (Salesforce.com Object Query Language)

A query language that allows you to construct simple but powerful query strings and to specify the criteria that should be used to select data from the Force.com database.

## SOSL (Salesforce.com Object Search Language)

A query language that allows you to perform text-based searches using the Force.com API.

## Source Report

A custom report scheduled to run and load data as records into a target object for an analytic snapshot.

## Standard Object

A built-in object included with the Force.com platform. You can also build custom objects to store information that is unique to your app.

**Subscriber**

The subscriber of a package is a Salesforce.com user with an installed package in their Salesforce.com organization.

**Summary Field**

Standard and custom summary fields defined for a report. Standard summary fields are report columns with one of the following summaries applied: sum, average, largest value, smallest value. Custom summary fields are user-defined custom summary formulas. In addition to showing summarized information, summary fields can be used to define charts and analytic snapshots.

**Summary Report**

Summary reports are similar to tabular reports, but also allow users to group rows of data, view subtotals, and create charts. They can be used as the source report for dashboard components. Use this type for a report to show subtotals based on the value of a particular field or when you want to create a hierarchical list, such as all opportunities for your team, subtotaled by Stage and Owner.

**Syndication Feeds**

Give users the ability to subscribe to changes within Force.com sites and receive updates in external news readers.

**System Administrator**

See Administrator (System Administrator).

**System Log**

A separate window console that can be used for debugging code snippets. Enter the code you want to test at the bottom of the window and click Execute. The body of the System Log displays system resource information, such as how long a line took to execute or how many database calls were made. If the code did not run to completion, the console also displays debugging information.

**System Testing**

The phase of testing that detects problems caused by multiple integrations or within the system as a whole. System testing should require no knowledge of the inner design of the code or logic.

**T****Tab**

A tab is an interface component that allows you to navigate around an app. A tab serves as the starting point for viewing, editing, and entering information for a particular object. When you click a tab at the top of the page, the corresponding tab home page for that object appears. A tab can be associated with an object, a Web page, or a Visualforce page.

### **Tabular Report**

Tabular reports are the simplest and fastest way to look at data. Similar to a spreadsheet, they consist simply of an ordered set of fields in columns, with each matching record listed in a row. Tabular reports are best for creating lists of records or a list with a single grand total. They can't be used to create groups of data or charts, and can't be used in dashboards unless rows are limited. Examples include contact mailing lists and activity reports.

### **Tag**

In Salesforce.com, a word or short phrases that users can associate with most records to describe and organize their data in a personalized way. Administrators can enable tags for accounts, activities, assets, campaigns, cases, contacts, contracts, dashboards, documents, events, leads, notes, opportunities, reports, solutions, tasks, and any custom objects (except relationship group members) Tags can also be accessed through the Force.com Web Services API.

In Salesforce CRM Content, a descriptive label that helps classify and organize content across workspaces. Users can view a list of all files or Web links that belong to a particular tag or filter search results based on a tag or tags.

### **Task**

Assigns a task to a user you specify. You can specify the Subject, Status, Priority, and Due Date of the task. Tasks are workflow and approval actions that are triggered by workflow rules or approval processes. For Calender-related tasks, see Activity (Calendar Events/Tasks).

### **Task Bar Links**

Links on tabbed pages that provide quick access to the most common operations available for a particular page, for example, creating a new account.

### **Test Case Coverage**

Test cases are the expected real-world scenarios in which your code will be used. Test cases are not actual unit tests, but are documents that specify what your unit tests should do. High test case coverage means that most or all of the real-world scenarios you have identified are implemented as unit tests. See also Code Coverage and Unit Test.

### **Test Drive**

A test drive is a fully functional Salesforce.com organization that contains an app and any sample records added by the publisher for a particular package. It allows users on AppExchange to experience an app as a read-only user using a familiar Salesforce.com interface.

**Test Method**

An Apex class method that verifies whether a particular piece of code is working properly. Test methods take no arguments, commit no data to the database, and can be executed by the `runTests()` system method either through the command line or in an Apex IDE, such as the Force.com IDE.

**Test Organization**

A Salesforce.com organization used strictly for testing. See also Sandbox Organization.

**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.

**Time-Dependent Workflow Action**

A workflow action that executes when the conditions of a workflow rule and an associated time trigger are met.

**Timeout**

See Session Timeout.

**Time Trigger**

An event that starts according to a specified time threshold, such as seven days before an opportunity close date. For example, you might define a time-based workflow action that sends email to the account manager when a scheduled milestone will occur in seven days.

**Translation Workbench**

Administration setup area where your users can translate custom field names, picklist values, record types, and page layout sections. The translation workbench also determines which users translate different languages.

**Trigger**

A piece of Apex that executes before or after records of a particular type are inserted, updated, or deleted from the database. Every trigger runs with a set of context variables that provide access to the records that caused the trigger to fire, and all triggers run in bulk mode—that is, they process several records at once, rather than just one record at a time.

**Trigger Context Variable**

Default variables that provide access to information about the trigger and the records that caused it to fire.

**U**

**Unit Test**

A unit is the smallest testable part of an application, usually a method. A unit test operates on that piece of code to make sure it works correctly.  
See also Test Method.

**Unlimited Edition**

Unlimited Edition is salesforce.com's flagship solution for maximizing CRM success and extending that success across the entire enterprise through the Force.com platform.

**Unmanaged Package**

A package that cannot be upgraded or controlled by its developer.

**Upgrading**

Upgrading a package is the process of installing a newer version. Salesforce.com supports upgrades for managed packages that are not beta.

**Uploading**

Uploading a package in Salesforce.com provides an installation URL so other users can install it. Uploading also makes your packaged available to be published on AppExchange.

**URL (Uniform Resource Locator)**

The global address of a website, document, or other resource on the Internet. For example, <http://www.salesforce.com>.

**URL S-Control**

An s-control that contains an external URL that hosts the HTML that should be rendered on a page. When saved this way, the HTML is hosted and run by an external website. URL s-controls are also called Web controls.

**User Acceptance Testing (UAT)**

A process used to confirm that the functionality meets the planned requirements. UAT is one of the final stages before deployment to production.

**User Interface**

The layouts that specify how a data model should be displayed.

**V****Validation Rule**

A rule that prevents a record from being saved if it does not meet the standards that are specified.

**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.com pages, or developers can associate their own logic with a controller written in Apex.

**Visualforce Controller**

See Controller, Visualforce.

**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****Web Direct Leads**

Web direct leads is a specific lead source indicating that the lead was generated when a user, who has bookmarked your website or directly typed the URL of your website into a browser, filled out the Web-to-Lead form containing the Salesforce.com tracking code.

**Web Control**

See URL S-Control.

**Web Links**

See Custom Links.

**Web Service**

A mechanism by which two applications can easily exchange data over the Internet, even if they run on different platforms, are written in different languages, or are geographically remote from each other.

**WebService Method**

An Apex class method or variable that can be used by external systems, such as an s-control or mash-up with a third-party application. Web service methods must be defined in a global class.

**Web Tab**

A custom tab that allows your users to use external websites from within the application.

**Wizard**

A user interface that leads a user through a complex task in multiple steps.

**Workflow and Approval Actions**

Workflow and approval actions consist of email alerts, tasks, field updates, and outbound messages that can be triggered by a workflow rule or approval process.

**Workflow Action**

An email alert, field update, outbound message, or task that fires when the conditions of a workflow rule are met.

**Workflow Email Alert**

A workflow action that sends an email when a workflow rule is triggered. Unlike workflow tasks, which can only be assigned to application users, workflow alerts can be sent to any user or contact, as long as they have a valid email address.

**Workflow Field Update**

A workflow action that changes the value of a particular field on a record when a workflow rule is triggered.

**Workflow Outbound Message**

A workflow action that sends data to an external Web service, such as another cloud computing application. Outbound messages are used primarily with composite apps.

**Workflow Queue**

A list of workflow actions that are scheduled to fire based on workflow rules that have one or more time-dependent workflow actions.

**Workflow Rule**

A workflow rule sets workflow actions into motion when its designated conditions are met. You can configure workflow actions to execute

immediately when a record meets the conditions in your workflow rule, or set time triggers that execute the workflow actions on a specific day.

### **Workflow Task**

A workflow action that assigns a task to an application user when a workflow rule is triggered.

### **Wrapper Class**

A class that abstracts common functions such as logging in, managing sessions, and querying and batching records. A wrapper class makes an integration more straightforward to develop and maintain, keeps program logic in one place, and affords easy reuse across components. Examples of wrapper classes in Salesforce.com include the AJAX Toolkit, which is a JavaScript wrapper around the Salesforce.com Web Services API, wrapper classes such as `CCritical Section` in the CTI Adapter for Salesforce CRM Call Center, or wrapper classes created as part of a client integration application that accesses Salesforce.com using the Force.com Web Services API.

### **WSDL (Web Services Description Language) File**

An XML file that describes the format of messages you send and receive from a Web service. Your development environment's SOAP client uses the Salesforce.com Enterprise WSDL or Partner WSDL to communicate with Salesforce.com using the Salesforce.com Web Services API.

## **X**

### **XML (Extensible Markup Language)**

A markup language that enables the sharing and transportation of structured data. All Force.com components that are retrieved or deployed through the Metadata API are represented by XML definitions.

## **Y**

No glossary items for this entry.

## **Z**

### **Zip File**

A data compression and archive format.

A collection of files retrieved or deployed by the Metadata API. See also Local Project.

# Index

## A

About this book 1  
 Accounts, mass updating accounts 143  
 Accounts, mass updating contacts 140  
 actionSupport tag 96  
 Adobe Flex 228  
 AJAX 228  
     in Visualforce pages 134  
     updating a page 88  
 AJAX Toolkit 13, 18  
 AJAX Tools 6  
 Alerts, workflow 10  
 Anonymous execution, Apex 12  
 anti-join 39  
     in Visualforce 110  
 Any array, partner WSDL 231  
 Apex 54  
     about 12  
     creating a button 66  
     custom settings 160  
     documentation 5  
     dynamic 40  
     integrating applications 227  
     mobile application, trigger for 204  
     preventing duplicate records 153  
     properties 136  
     triggers 142, 149, 152, 153, 159  
     WSDL 229  
 API  
     batching records 239  
     client applications 255  
     documentation 5  
     integrating applications 227  
     introduction 13  
     login() call 233  
     metadata introduction 14  
     queries and searches 16  
     query() 238  
     queryMore() 238  
     session ID 233  
     setting the session timeout value 237  
     setting up 228  
     wrapper classes 242, 258  
 API Only permission 228

Approval processes 10, 54  
     else option 50  
     parallel approvers 50  
 Approvals process  
     managing 44  
 Architecture, client application 257  
 Asynchronous outbound messages 56  
     tracking 59  
 Attributes, SObject 18, 24  
 Authenticating calls to the API 233  
 Averages, roll-up summary fields 121

## B

Batch Apex 143  
 Batching records for API calls 239  
 Bulk API 14  
     documentation 6  
 Bulk processing, trigger 142, 153  
 Buttons  
     creating with Apex 66  
     overriding with Visualforce 64

## C

C#.NET 228  
     using the partner WSDL 231  
 Certificates, client 230  
 Child records  
     creation from parent 159  
     mass updating 140  
 Classes, Apex 12  
 Client applications 255  
     configuration files 260  
     SOAP message compression 260  
 Client certificates 230  
 Client classes, API 242, 258  
 commandAction tag, Visualforce 71  
 commandButton tag, Visualforce 80  
 commandLink tag, Visualforce 110, 134  
 composition tag 98  
 Compression, SOAP message 260  
 Concatenate 123  
 Configuration files 260

- Console 11  
 Contacts  
     finding records 29  
     mass updating 140  
 Controllers, Visualforce  
     creating dependent controllers and pages 73  
     query string parameters 132  
 COUNT() 33  
 Cross-object formulas 123  
 CSS 104  
 Currency field format 68  
 Custom buttons  
     creating with Apex 66  
 Custom component 82  
 Custom fields  
     about 9  
     encrypted fields 118  
     history tracking 10  
 Custom help 81  
 Custom junction objects 114  
 Custom links, passing UTF-8 characters into 177  
 Custom list controller 110  
 Custom objects  
     about 9  
     creating 118  
 Custom relationships 9  
 Custom report types 11  
 Custom settings 160  
 Custom views 11
- D**
- Dashboards 11  
 Data actions 112  
 Data display 112  
 Data model  
     creating ERD diagrams 24  
     examining with enterprise WSDL and AJAX Toolkit 18  
     examining with SoqlXplorer 24  
 dataList tag, Visualforce 110  
 dataTable tag, Visualforce 69  
 Date literals, relative 31  
 Default field values 10  
 define tag 98  
 Delegated authentication 193, 216  
 describeGlobal() 18  
 Development languages, supported 228  
 Divisions, SOSL filtering based on 32  
 Duplicate records, preventing 153
- Dynamic Apex 40  
 Dynamic Approval Routing application 54  
 Dynamic approvals 54  
 Dynamic page updates 88
- E**
- Email  
 alerts, workflow 10  
 attachments 197  
 creating 193  
 creating records 196  
 example code 202  
 inbound 194, 196, 197  
 messaging 10  
 outbound 198  
 receiving 193  
 retrieving information 194  
 sending 193  
 services, activating 194  
 templates 198  
 encodeURI() JavaScript function 177  
 Encrypted fields 118  
 Enterprise WSDL 229  
 ERD diagrams, creating 24  
 Errata, book 7  
 External IDs and upserting 228
- F**
- facet tag, Visualforce 80, 134  
 Feedback, sending book 7  
 Feeds 186  
 Fields  
     custom 9  
     default values 10  
     formula 10, 121  
     history tracking 10  
     roll-up summary 121  
 FieldsToNull array, artner WSDL 231  
 Filtering using Divisions in SOSL 32  
 Finding records 29  
 Force.com IDE 6, 14  
 Force.com Migration Tool 6, 14  
 Force.com Platform  
     documentation 4  
     point-and-click setup tools 9  
     understanding 7, 9  
 Force.com sites  
     See Also Sites 184

## Index

form tag, Visualforce 71, 80, 172  
Formatting currencies 68  
Forms, building in Visualforce pages 71  
Formula fields 10  
Formulas, cross-object 123  
Frontdoor URLs 18  
Functions  
    HYPERLINK 123  
    REGEX() 118  
    VLOOKUP() 129

## G

Global objects variable 132

## H

Help with this book 1  
Help, field-level 118  
Hierarchy, role 10  
HTML in Visualforce 70  
HYPERLINK 123

## I

ID prefixes, record 18  
ID, session 233  
IDE 14  
image tag, Visualforce 172  
include tag 96  
Input components, Visualforce 71, 172  
inputCheckbox tag, Visualforce 71  
inputField tag, Visualforce 71, 80  
inputHidden tag, Visualforce 71  
inputSecret tag, Visualforce 71  
inputText tag, Visualforce 71  
inputTextarea tag, Visualforce 71, 172  
insert tag 98  
Instance, Salesforce.com 234  
Integrating with Salesforce Mobile 193  
Integration users 228  
Iteration components, Visualforce 69

## J

Java 228  
    batching records 239  
    query/queryMore pattern implementation 238  
    using the partner WSDL 231

Java (*continued*)  
    wrapper class implementation 243  
JavaScript encodeURI() 177

## L

Languages, supported development 228  
Layouts, page 11  
Leads, finding records 29  
LIMIT keyword, SOQL 33  
Literals, relative date 31  
Locking records 126  
login() call, API 233

## M

Mac OS X, SoqlXplorer 24  
Manual sharing rules 10  
Many-to-many relationships 114  
Mass updating records 140, 143  
Matrix-based approvals 54  
MessageElement, partner WSDL 231  
Messages, asynchronous outbound 56  
    tracking 59  
Messaging 193  
Metadata API  
    about 14  
    documentation 5  
Metadata WSDL 229  
Migration Tool 14  
Mobile 13  
    Apex trigger 204  
    Visualforce Mobile page, creating 211  
Modify All Data permission 228  
MVC design paradigm 257

## O

Objects  
    about 9  
    attributes 18, 24  
    frontdoor URLs 18  
    ID prefixes 18  
Operators, concatenate 123  
Organization-wide defaults 10  
Outbound messages, asynchronous 56  
    tracking 59  
Outer joins in SOQL 39  
outputLabel tag, Visualforce 80  
outputLink tag, Visualforce 132

outputPanel tag, Visualforce 80

## P

Page layouts 11

pageReference class, Visualforce 80

Pages, overriding with Visualforce 85

panelGrid tag, Visualforce 80

panelGroup tag, Visualforce 80

param tag, Visualforce 132

Partner WSDL 229, 231

Password Never Expires permission 228

Perl 228

Permissions

- about 10

- API Only 228

- Modify All Data 228

- Password Never Expires 228

Person accounts, finding records 29

PHP 228

- batching records 239

Point-and-click setup tools 9

Prefixes, record ID 18

Previewing query results 33

Processes, approval 10

Profiles, user 10

Python 228

## Q

Queries, previewing results 33

Query string parameters, Visualforce 132

query() 26

query/queryMore pattern 238

## R

Records

- bulk processing in a trigger 142

- ID prefixes 18

- locking 126

- preventing duplicates with Apex 153

Recruiting app, sample 3

Recursive triggers, controlling 149

Redirects, overriding with Visualforce 87

REGEX() function 118

Related records, fields 123

relatedList tag, Visualforce 132, 134

Relationships

- about 9

Relationships (*continued*)

- many-to-many 114

- traversing in SOQL 27

- viewing SObject 18, 24

Relative date literals 31

Reports 11

reRender attribute, Visualforce 134

Role hierarchy 10

Roll-up summary fields 121

- in cross-object formulas 126

Ruby on Rails 228

Rules

- sharing 10

- validation 10

- workflow 10

runAs method 170

## S

Salesforce Mobile 13

Salesforce.com instance 234

SAML 216

search() 26

Security settings 10

selectOption tag, Visualforce 172

selectRadio tag, Visualforce 172

semi-join 39

Session ID, API 233

Sessions

- managing 236

- setting the timeout value 237

Sharing rules 10

Single sign-on 193, 216

- clients 222

Sitemaps, creating for sites 188

Sites

- about 13

- branding 184

- CNAME 180

- components 184

- custom domain 180

- feeds 186

- merge fields 182

- site template 184

- sitemaps 188

- static resources 184

- Visualforce 182

- Web-to-Lead form 190

SOAP clients 229

SOAP message compression 260

## Index

- SObjects
    - attributes 18, 24
    - frontdoor URLs 18
    - ID prefixes 18
  - SOQL
    - anti-join 110
    - COUNT() 33
    - LIMIT 33
    - outer joins 39
    - previewing results 33
    - relative date literals 31
    - semi-join 110
    - sorting results 34
    - SOSL, compared to 26
    - traversing relationships 27
  - SqlXplorer
    - Mac OS X 6
    - using 24
  - Sorting query results 34
  - SOSL
    - filtering based on division 32
    - relative date literals 31
    - SOQL, compared to 26
    - sorting results 34
    - WHERE clause 32
    - WITH clause 32
  - Standard list controller 107
  - Static resources 67
  - Styles in Visualforce 70
  - System global object, Visualforce 132
- ## T
- Tabbed accounts 102
  - Tabs 11
  - Tags
    - adding 261, 263
    - removing 261, 263
    - updating 265
    - viewing 37
    - viewing records 38
  - Tasks, workflow 10
  - Testing Apex 170
  - Timeout values, session 237
  - Tools, integrated 6
  - Trigger.new 152
  - Trigger.old 152
  - Triggers 159
    - about 12
    - bulk processing 142, 153
- Triggers (*continued*)
    - comparing Trigger.old and Trigger.new 152
    - preventing duplicate records 153
    - recursive 149
- ## U
- Upsert with external IDs 228
  - URL query string parameters, Visualforce 132
  - URLs, frontdoor 18
  - Users
    - about profiles 10
    - integration 228
  - Using this book 1
  - UTF-8 characters, passing into custom links 177
- ## V
- Validation rules 10, 118, 126
  - VLOOKUP() 129
  - VB.NET 228
    - using the partner WSDL 231
    - wrapper class implementation 250
  - Verifying data 129
  - Views, custom 11
  - Visualforce 81
    - about 12
    - adding feeds 186
    - AJAX 134
    - Apex properties 136
    - apex:outputLink tag 132
    - apex:param tag 132
    - commandAction tag 71
    - commandButton tag 80
    - commandLink tag 110, 134
    - conditional overrides 93
    - controller extension 93, 110
    - creating dependent controllers and pages 73
    - creating forms 71, 80
    - CSS 104
      - custom component 82
      - custom list controller 110
    - dataList tag 110
    - dataTable tag 69
    - documentation 5
    - dynamically updating 88
    - editing multiple records 107
    - email template 198
    - excluding records 110
    - facet tag 80, 134

Visualforce (*continued*)  
 form tag 71, 80, 172  
 formatting currencies 68  
 formatting dates 68  
 Google Charts, integrating with 172  
 HTML tags 70  
 image tag 172  
 include templates 96  
 including records 110  
 inputCheckbox tag 71  
 inputField tag 71, 80  
 inputHidden tag 71  
 inputSecret tag 71  
 inputText tag 71  
 inputTextarea tag 71, 172  
 iteration components 69  
 mobile page, creating 211  
 multiple sObjects on a page 136  
 navigation 72  
 outputLabel tag 80  
 outputPanel tag 80  
 overriding buttons 64  
 overriding pages 85  
 pageReference class 80  
 panelGrid tag 80  
 panelGroup tag 80  
 queries 136  
 query string parameters 132  
 redirecting pages 87  
 relatedList tag 132, 134  
 reRender attribute 134  
 selectOption tag 172  
 selectRadio tag 172  
 skeleton templates 98  
 standard list controller 107  
 static resources 67  
 styles 70  
 System global object 132  
 tabs 102  
 wizards 72

VLOOKUP() function 129

## W

Web controls  
 SOAP message compression 260  
 Web services  
 comparing Salesforce.com WSDLs 229  
 generating client certificates 230  
 generating Salesforce.com WSDLs 230  
 sending asynchronous outbound messages 56  
 SOAP message compression 260  
 tracking asynchronous outbound messages 59  
 Web-to-Lead form, creating for sites 190  
 WHERE clause, SOSL/SOQL 32  
 WITH clause, SOSL 32  
 Wizards, creating with Visualforce pages 72  
 Workflow  
 asynchronous outbound messages 56, 59  
 email alerts 45, 47, 48, 60  
 field updates 49  
 formulas 47, 49  
 managing 44  
 rules 10, 45, 47, 48, 49, 60  
 time triggers 45, 60  
 Wrapper classes, API 242, 257, 258  
 WSDLs  
 comparing 229  
 generating 230  
 using the partner WSDL 231

## X

XMLElement, partner WSDL 231

## Z

ZIP codes 129