Creating On-Demand Applications

An Introduction to the Force.com Platform

By Caroline Roth
Dave Carroll
Nick Tran

With contributions by
Adam Gross
Andrea Leszek

Creating On-Demand Applications

ISBN: 978-0-9789639-1-0

With special thanks to Steve Anderson, Mysti Berry, Eric Bezar, Steve Fisher, Ron Hess, Paul Kopacki, Mary Scotton, Andrew Smith, Tom Tobin, Adam Torman, and Sarah Whitlock.

Table of Contents

Preface

As users of the Internet, we're all familiar with the fascinating, innovative, creative, and sometimes silly ways in which it has changed how we work and how we play. Much of that excitement comes from the fact that how we now create and share content is as different from the "traditional" publishing models as how we now consume it. From websites to wikis to blogs, and more, it's exciting to watch the innovations taking place that are changing how we communicate and collaborate.

While these changes have certainly impacted how we work with content, a similar set of Internet-driven ideas and technologies is changing how we build and work with business applications. While yesterday's business applications required thousands, if not millions, of dollars and sometimes years of professional services help to set up and customize, the technologies offered by the Internet today make it much easier to create, configure, and use business applications of all kinds. Indeed, the power of the Internet has given us the ability to solve new kinds of business problems that, because of complexity or cost, had previously remained out of reach.

Just as the changes that moved publishing technology from paper to bits made it possible for us to have information about anything in the whole world right at our fingertips, the changes in application technology make it similarly possible to imagine a robust, enterprise-class application for almost any business need. Sound pretty good? Then you're probably wondering: "What's the catch? What's the magic that makes this possible?"

Welcome to On-Demand Computing

These new ways of building and running applications are enabled by the world of *on-demand computing*, where you access applications, or *apps*, over the Internet as utilities, rather than as pieces of software running on your desktop or in the server room. This model is already quite common for consumer apps like email and photo sharing, and for certain business applications, like customer relationship management (CRM).

Because almost all apps these days are delivered via a Web browser, it's increasingly hard to tell which applications are "traditional software," and which are being run on demand. As with the Internet, on-demand applications have grown so ubiquitous that almost every business user interacts with at least one, whether it's an email service, a Web conferencing application, or a sales system.

On-Demand Platforms

A new twist, the *on-demand platform*, is making the delivery of application functionality even more interesting. Increasingly, on-demand applications are starting to look less like websites and more like platforms, meaning they are starting to sprout Application Programming Interfaces (APIs), code libraries, and even programming models. Collectively, these new kinds of development technologies can be thought of as on-demand platforms.

Similar to traditional platforms, on-demand platforms provide tools that allow developers to leverage existing functionality to create something new. However, because on-demand platform tools are accessed freely over the Internet rather than through an operating system or package that was installed on a local machine, developers don't need to worry about the logistics of putting together an executable that will be installed on a user's machine. Anyone with a Web browser can access it!

The possibilities presented by this new type of platform have emerged quickly, spurred on by the popularity of *mash-ups*—a website or application that combines tools from multiple on-demand platforms to create new functionality. Some of the on-demand platform tools used in today's mash-ups include innovations like Google's search API, which allows developers to use the power of that search engine in their applications, eBay's APIs for auctions and listings, or Amazon.com's system for creating entirely new storefronts. For example, almost any real estate website or application these days uses Google or Yahoo! maps under the hood, illustrating how these new APIs are now commonly running alongside the more traditional database, app server, or operating system platforms.

The Programmable Web

The new ways you can plug into websites are exciting and essential, but they're really just pieces of the much larger picture of on-demand platforms, or what people are now calling the *programmable Web*. Not only can you reuse or mash up on-demand platform tools in new and interesting ways, but these tools themselves are becoming customizable and programmable. In a way, this is the database analogue to My Yahoo's content model—just as you can easily configure a page to display only the headlines or feeds that you find most interesting, so you

can now modify the structure and behavior of an on-demand business application, or create a new one, using just a browser.

About This Book

This book introduces you to Force.com, salesforce.com's on-demand platform for building and running business applications that represents one of the most interesting aspects of the programmable Web today.

To illustrate the technologies available on the Force.com platform, and to show you just how easy it is to create your own business application with the platform, this book walks you through the process of creating a new on-demand application for a recruiting department. To follow along you won't need to learn any programming languages or hack your way through cryptic configuration documents—instead, you'll just need to point-and-click your way through the Salesforce Web interface, following the easy step-by-step instructions in the book.

Note: An online version of this book is available on the Apex Developer Network website at wiki.apexdevnet.com/index.php/Creating_On-Demand_Apps.

Intended Audience

This book focuses primarily on the native capabilities of the Force.com platform and can be easily understood by anyone from a business user to a professional developer. To get the most out of the book, you should be familiar with the salesforce.com CRM application, and have a general idea of basic database concepts such as tables and fields.

In the final chapter, the book introduces the power of composite apps—that is, Force.com platform apps that leverage services from other websites. To fully understand that chapter, you should be familiar with HTML and JavaScript. However, even if you're not an experienced developer, you can still follow along to gain a deeper understanding of what can be done with Force.com.

Chapter Contents

If you're already familiar with the Force.com platform, you can skip around to the chapters in which you're most interested:

Learn about the technologies behind Force.com, including the AppExchange directory.

- *Chapter 2: About the Sample Recruiting App* on page 15

  Learn about the recruiting application that we'll be building in this book and the fictitious company for whom we'll be building it.

- *Chapter 3: Reviewing Database Concepts* on page 23

  Review database concepts such as tables, records, fields, keys, and relationships.

- *Chapter 4: Building a Simple App* on page 31

  Create the first custom object in our recruiting app, and add several basic fields.

- *Chapter 5: Enhancing the Simple App with Advanced Fields and Page Layouts* on page 55

  Add picklists, dependent picklists, and formula fields to the custom object, and then edit the layout of the object's detail page.

- *Chapter 6: Expanding the Simple App Using Relationships* on page 81

  Add three more custom objects to our recruiting app, and associate them with one another using lookup and many-to-many relationships.

- *Chapter 7: Securing and Sharing Data* on page 109

  Set up rules for who can read, create, edit, and delete records in the app.

- *Chapter 8: Using Custom Workflow and Approval Processes* on page 153

  Define workflow rules and approval processes that assign tasks, update fields, and send emails when certain criteria are met.

- *Chapter 9: Analyzing Data with Reports and Dashboards* on page 193

  Create custom reports, charts, and dashboards that give users a bird's-eye view of recruiting data.

- *Chapter 10: Moving Beyond Native Apps* on page 225

  Learn about the Force.com Web Services API and how it can be used with s-controls to create a mash-up of the platform with Yahoo! maps.

- *Chapter 11: Learning More* on page 243

  Find out where you can get more information about developing on the platform.

- *Glossary* on page 249

  Look up the definition of any term you find unfamiliar.

Note: This book contains lots of screenshots. Because Force.com is a rapidly developing platform, the screenshots might vary slightly from what you see on the screen, but don't worry! These differences should be minor and won't affect your understanding of the system.

Signing Up for a Developer Edition Account

To follow along with the exercises in this book, go to the Apex Developer Network at www.salesforce.com/developer/ and click **Join Now**. You'll get a free, fully-functional Developer Edition account, and you'll also become part of the growing community of Force.com developers around the world.

Sending Feedback

Questions or comments about anything you see in this book? Suggestions for topics that you'd like to see covered in future versions? Go to the Apex Developer Network discussion boards at community.salesforce.com/sforce?category.id=developers and let us know what you think! Or email us directly at adn@salesforce.com.

About the Apex Developer Network

The Apex Developer Network (ADN) at www.salesforce.com/developer/ is a community of developers who customize and build on-demand applications with the platform. ADN members have access to a full range of resources, including sample code, toolkits, an online developer community, and the test environments necessary for building apps. The ADN website includes an online version of this book (written by ADN staff members) and has information about the ADN@Dreamforce event that we hold every year for Force.com developers. If you need more info, have a question to ask, are seeking a toolkit or sample, or just want to dig a little deeper into Force.com development, ADN is where it all comes together.

To find out more about the resources available on the ADN website, see *The Apex Developer Network* on page 244.

Chapter 1

Introducing the Force.com Platform

In this chapter ...

Force.com is an on-demand platform offered by salesforce.com that makes building, sharing, and running business applications simpler and more powerful than has previously been possible.

To best understand the kinds of business apps you can build with the platform, we'll first briefly examine the basic components of the most popular application for the platform, salesforce.com's sales automation app, Salesforce SFA. We'll then highlight some of the key technologies that differentiate Force.com from other platforms you may have already used.

The Basics of an App

If you haven't used Salesforce before, you'll find it worthwhile to spend a bit of time clicking around the basic Salesforce SFA app included with every Salesforce edition. For example, try using the Developer Edition account you should have already signed up for to create a contact or account. The interface for these tasks has a lot in common with the interface of whatever app you're planning to build.

Note: Haven't signed up for a Developer Edition account yet? Go to the ADN website at www.salesforce.com/developer/ and click **Join Now!**.

Tabs

As you can see when you start clicking around, there are a few key elements that form the foundation of the Sales Automation app and of most applications created with the platform. First, across the top of the app is a set of *tabs* that segment the app into different parts. Each tab corresponds to a type of object, such as an account or contact, and within a tab you can perform actions on particular records of that tab's type. For example, when you click on the Accounts tab, you can create a new record for the "Acme" account. You can also edit existing accounts, or filter lists of accounts by certain criteria. Most app development work revolves around creating tabs and defining the data and behaviors that support them.

Forms

A second key element is the *form* that is displayed as part of a tab. As in any business app, forms are the primary means of entering and viewing information in the system. Forms allow you to view and edit the data associated with a particular record on a tab, like the contact "Jerome Garcia" on the Contacts tab. When developing a new app you can define what information appears in each form, and how it is organized. For example, the form for a contact record includes fields such as `Last Name`, `Home Phone`, `Mailing City`, `Title`, `Birthdate`, `Reports To`, and `Account`. In a Force.com app, the form used to enter information is referred to as an *edit page* and the read-only view of that information is referred to as a *detail page*.

Links

Finally, because Force.com apps are delivered via a Web browser, they use *links* to provide navigation to related data. For example, on an account detail page, there are links to related records, such as the contacts who belong to the account and the sales user who manages the

account. Other links take you to recently visited records and to areas of the app where users can set personal preferences. These links provide navigation both within an app and out into the Web.

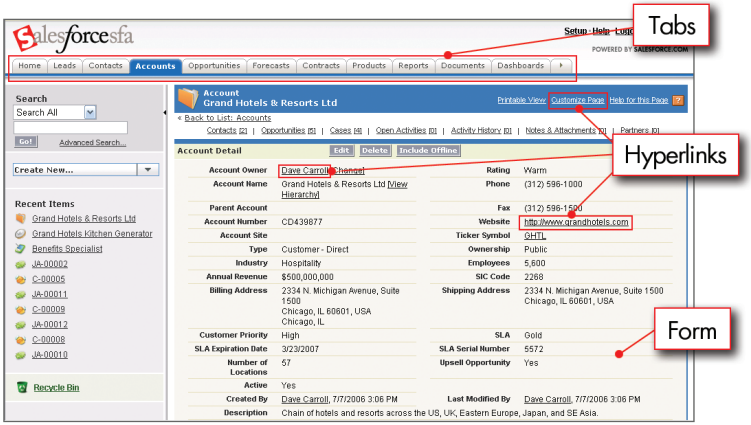


Figure 1: Force.com Apps Include Tabs, Detail Pages, and Links

The Benefits of a Force.com App

To better understand what the platform is best suited for, let's look beyond the core elements of tabs, forms, and links, and into the types of applications they enable. Two huge benefits start to come into focus when you look across Force.com apps in general: they're both data-centric and collaborative.

Data-Centric Apps

Because the platform is centered around a database, it allows you to write apps that are *data-centric*. A data-centric app is an application that is based on structured, consistent information such as you might find in a database or an XML file. We can find these data-centric apps everywhere, in small desktop databases like Microsoft Access or FileMaker, all the way to the huge systems running on database management systems like Oracle or MySQL. Unlike applications that are built around unstructured data, like plain text documents or HTML files, data-centric apps make it easy to control, access, and manage data.

For example, consider an exercise such as trying to determine the total sales for a month from a set of Microsoft Word-based contracts versus a set of contracts in a simple database. Whereas it takes a lot of effort to open each Word document, find the contract total, and then add them

all together, if this data is stored in the database of a data-centric app, we can more efficiently get the same result by issuing a single query.

While a normal person usually doesn't need a data-centric application to keep track of anything other than contacts, photos, or possibly music, companies of all sizes constantly need to query and aggregate their large amounts of data to make fast business decisions. As a result, the data-centric nature of Force.com makes it the perfect platform to build and host business applications.

Collaborative Apps

Because the platform can be accessed by multiple users at the same time, it also allows you to write apps that are *collaborative*. A collaborative app is an application with data and services that are shared by multiple users in different locations. Unlike more traditional forms of software that are installed on a single machine and are hard to access from a distance, collaborative apps on the platform can be accessed from anywhere in the world with only a Web browser. This makes it easy for teams to work together on activities like selling a product, managing a project, or hiring an employee.

In addition to easy access over a Web browser, a number of built-in platform features also facilitate productive group collaboration:

- The platform's security and sharing model allows you to finely control a user's access to different data
- Workflow rules allow you to automatically assign tasks, update data, or send email alerts when certain business events occur, such as the creation of a new record or a change in the value of a record field
- Approval processes allow you to set up a sequence of steps necessary for a record to be approved, including who must approve it at each step

Collectively, these features provide a framework for sharing apps across groups, divisions, and entire corporations without relinquishing administrative control over sensitive data.

The Technologies Behind a Force.com App

Now that we've talked about the kinds of apps the platform can build, let's review some of the technologies behind the platform itself. These technologies have a big impact on what the platform supports, and what it's like to develop on it.

Table 1: Key Technologies Behind the Platform

Technology	Description
Multitenant architecture	An application model in which all users and apps share a single, common infrastructure and code base
Metadata-driven development model	An app development model that allows apps to be defined as declarative "blueprints," with no code required. Data models, objects, forms, workflows, and more are defined by metadata.
Force.com Web Services API	An application programming interface that defines a Web service that provides direct access to all data stored in Force.com from virtually any programming language and platform
AppExchange directory	A Web directory where hundreds of AppExchange apps are available to Salesforce customers to review, demo, comment upon, and/or install. Developers can submit their apps for listing on the AppExchange directory if they want to share them with the community.

A Multitenant Architecture

The platform's *multitenant architecture* means that all users share the same physical instance and version of any application that runs on it. In contrast to their single-tenant counterparts, such as client-server enterprise applications or email servers, multitenant applications are designed so that any upgrades to the platform or the apps it supports happen automatically for all users at once. Consequently, no one has to worry about buying and maintaining their own physical stack of hardware and software, or making sure that their applications always have the latest patch installed.

Besides the Force.com platform, several popular, consumer-based applications also use a multitenant architecture, including eBay, My Yahoo!, and Google Mail. Multitenant architecture allows these applications to be low cost, quick to deploy, and open to rapid innovation—exactly the qualities for which salesforce.com has also become known.

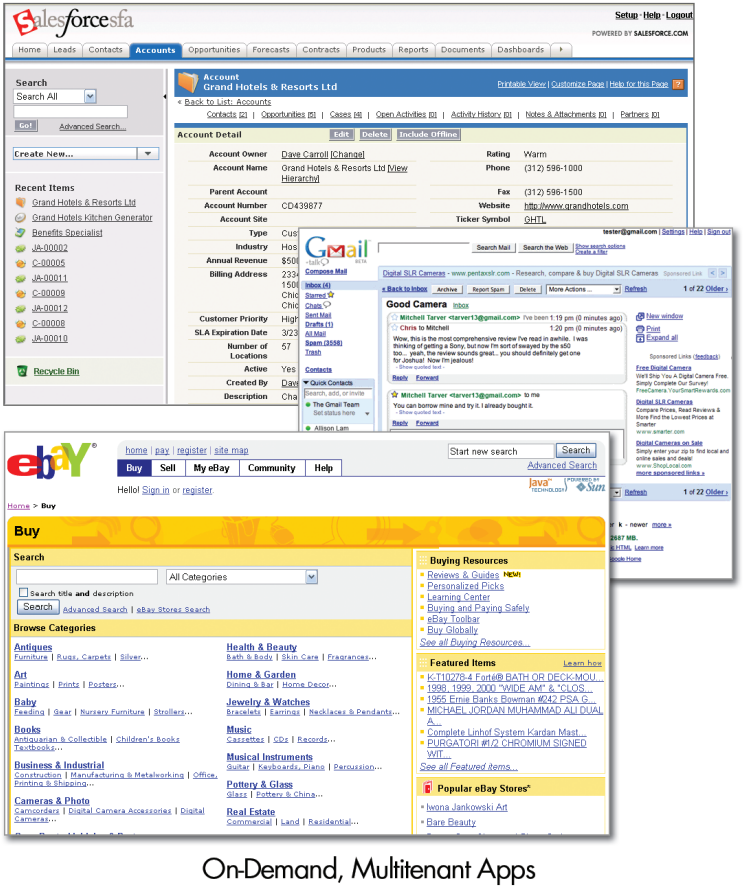


Figure 2: On-Demand, Multitenant Applications

The platform's multitenant architecture also impacts how developers use the platform to create new applications. Specifically, it defines a clear boundary between the platform and the applications that run on it. A boundary is important because it allows applications to define their own components without jeopardizing the functionality of the core platform or the data stored by other users.

A Metadata-Driven Development Model

Force.com also uses a *metadata-driven development model* to help app developers become more productive in putting together basic apps. It means that the basic functionality of an app—that is, the tabs, forms, and links—are defined as metadata in a database rather than being

hard-coded in a programming language. When a user accesses an app through the Force.com platform, it renders the app's metadata into the interface the user experiences.

As a result of metadata-driven development, Force.com app developers work at a much higher level of abstraction than if they developed applications using Java or C#, and are shielded from having to worry about low-level system details that the platform handles automatically. At the same time, Force.com developers can also leverage advanced features that the platform provides by default.

Although at first glance metadata-driven development may seem somewhat esoteric, it's exactly the same model for how Web browsers work. Instead of hard coding the definition of a Web page in a free-form programming language, a Web page author first defines the page as HTML, which is itself a kind of metadata. When the page is requested by a user, the Web browser renders the page using the metadata provided in the HTML tags. Even though the HTML/browser combination does not allow authors as much formatting power as they might get in a regular publishing tool, it simplifies the work of publishing content to a wide audience and increases the Web page author's overall productivity.

Likewise, even though Force.com also does not allow app developers as much power as they might get from free-form coding in Java or .NET, it vastly simplifies the work of building an app and increases a developer's overall productivity. And, like Web pages that use JavaScript or Flash to add functionality to HTML pages, Force.com also provides ways for more advanced developers to add third-party functionality to native platform apps—something we'll talk about next.

The Force.com Web Services API

The platform's metadata-driven development model allows app developers to quickly build a lot of *native* functionality—that is, functionality that's understood by the platform and that can be defined in metadata. However, sometimes app developers need an app to do something that isn't available natively on the platform. They want to go beyond the metadata model and use traditional programming to create behaviors that fall outside of the platform's constraints. To do this, they can use the *Force.com Web Services API*.

The API provides a straightforward, powerful, and open way to programmatically access the data and capabilities of any app running on the platform. It allows programmers to access and manipulate apps from any server location, using any programming language that supports Web services, like Java, PHP, C#, or .NET. Because Web services are—not surprisingly—based on Web standards, they're well suited to traverse firewalls and leverage the rest of the Internet infrastructure already in place.

While the API opens the door to the power of traditional programming languages, using it also typically requires a lot more work for app developers. As a result, most apps that leverage the API to create advanced functionality do so in a way that builds upon the native Force.com capabilities. For this reason, apps that leverage the API are known as *composite* apps.

The AppExchange Directory

The final piece of technology that differentiates the Force.com platform from other platforms is the *AppExchange*. The AppExchange is a Web directory where apps built on the Force.com platform are available to salesforce.com customers to browse, demo, review, and install. Developers can submit their apps for listing on the AppExchange directory if they want to share them with the community.

To fully appreciate the benefits of the AppExchange, take a quick tour at www.salesforce.com/appexchange/. There you'll see the hundreds of innovative and exciting apps that exist today, including everything from payroll management to telephony integration, service and support surveys, adoption dashboards, and beyond. Some of these apps have been created in-house at salesforce.com, but most are built by partners and individual developers who have chosen to take advantage of the Force.com platform.

Chapter 2

About the Sample Recruiting App

In this chapter ...

The goal of this book is to show you how easy it is to create powerful, multifaceted applications that solve common business problems. To do so, let's walk through the steps of creating a simple application for a make-believe company called Universal Containers.

Like many companies that have grown rapidly, Universal Containers has been experiencing a few growing pains, especially in its Human Resources department. In this book, we're going to build a Recruiting app for the company that allows it to move away from the Microsoft Word documents and Microsoft Excel spreadsheets that it has traditionally used to an application that's available on demand.

By the time we finish building the Recruiting app in this book, you should feel confident enough to build a custom on-demand application that suits your own company's needs. So let's get started!

Note: It's not the intent of this book to provide you with a full-featured Recruiting application that meets all of your business needs. Rather, we'll be using this Recruiting app as a jumping-off point for learning about what's generally included in an app.

If you do want a full-featured Recruiting app, you can either build your own using the app that we describe in this book as a good starting point, or you can download a pre-built app from the Recruiting section of the AppExchange directory at www.salesforce.com/appexchange/.

About Universal Containers

First, let's learn a little more about our fictional company, Universal Containers.

Universal Containers is a rapidly growing international supplier of container products. The company produces every kind of container from simple overnight letter mailers to custom equipment packaging to large cargo shipping containers. In addition, Universal Containers develops and maintains its own proprietary software to facilitate the design of its various types of containers. As such, Universal Containers has a very diverse group of employees, including facilities and operations professionals, software and design engineers, financial accountants, and legal and human resources personnel.

Historically, the Human Resources department has used Microsoft Word documents and Microsoft Excel spreadsheets to manage the recruiting and hiring process for new employees. However, over the last two quarters it's become evident that unless this process is replaced by one that is more collaborative, reliable, and scalable, the department won't be able to meet its hiring goals for this fiscal year. Universal Containers needs a centralized application that can bring all of its recruiting and hiring processes together, and the company has hired us to solve this problem. Our approach will be to leverage their Salesforce account and build a recruiting application on the Force.com platform. We're going to introduce Universal Containers to the world of on-demand, custom apps!

Considerations for the Recruiting App

After meeting with Megan Smith, Universal Container's vice president of Human Resources, we've drawn up a few requirements for the new Recruiting app. The app needs to:

- Track positions in all stages of the process, from those that are open to those that have been filled or canceled
- Track all of the candidates who apply for a particular position, including the status of their application (whether they've had a phone screen, are scheduled for interviews, have been rejected or hired, or have passed on an offer that was presented)
- Allow employees to post reviews for candidates whom they've interviewed
- Provide security for the recruiting data so that it's not mistakenly viewed, edited, or deleted by employees who shouldn't have access
- Automatically inform the relevant recruiter about the next steps that should be taken when a decision has been made about an applicant
- Automatically inform all employees of new positions that have been posted
- Make sure that a new job opening has executive approval before it becomes active
- Include reports that give users an overview of recruiting status

- Allow recruiters to map the locations of all candidates who are applying for a position, to better understand relocation expenses

Though modest, an app that includes this functionality is going to greatly increase the efficiency of Universal Containers' recruiting and hiring processes. Let's now take a look at how we'll implement these requirements.

Building the App: Our Design

As application developers, our job now is to figure out which platform components are going to allow us to build all of these features.

One way to split up the work is to look at which requirements can be implemented by using just the point-and-click tools of the platform and which requirements must be implemented by leveraging other Web applications. The former method, which uses *native components*, is typically fast and simple to use, requiring only point-and-click setup rather than more traditional coding. The latter method, which uses *composite components*, give us more control and flexibility in what we do but requires more work.

The following diagram shows how features implemented with each method are created, split out by their user interface layer, business logic, and data model.

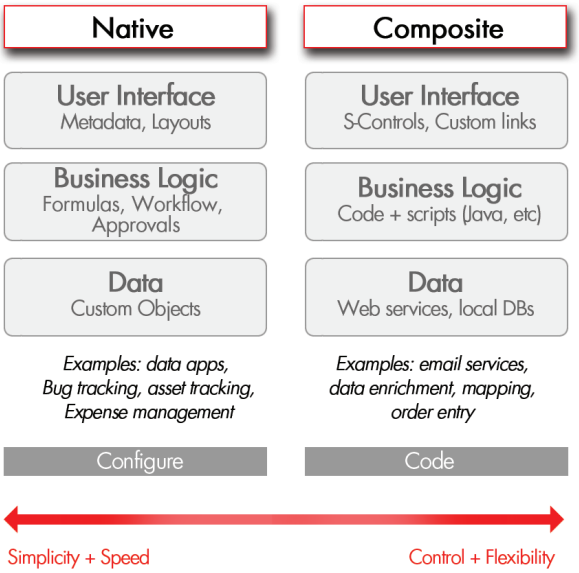


Figure 3: Native Versus Composite Components

Fortunately, all but one of our Recruiting app's requirements can be implemented using the native component method—an example of why the platform is such a powerful development environment! Now let's go into a little more detail about what we're going to build.

Native Components

Natively, there are several platform components that are going to help us implement our Recruiting app requirements. These include:

- Custom objects
- Security and sharing rules
- Workflow and approval processes
- Custom reports and dashboards

We're going to learn about all of these things in a lot more detail in later chapters, but for now, let's get a quick preview of what's in store.

Custom Objects

Custom objects are the native components that model the data we need to store in our Recruiting app. Similar to a database table, a custom object is composed of several fields that store information such as a job applicant's name, or the maximum salary for a particular position. However, unlike traditional database tables, we don't need to write any SQL in order to create custom objects. We can simply point and click in the platform to create as many objects as we need.

For our Recruiting app, we'll be creating four custom objects to track recruiting-related data:

- Position
- Candidate
- Job Application
- Review

Three of these objects, Candidate, Position, and Job Application, will be displayed as tabs in our application. When a user clicks one of the tabs, he or she will have access to individual instances of that particular object, as shown in the following screenshot.

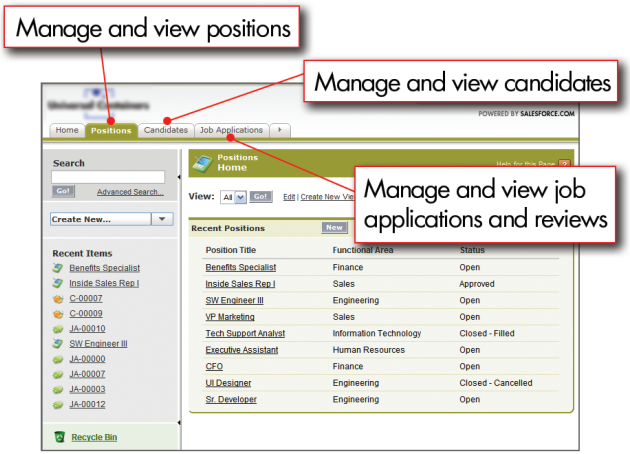


Figure 4: Recruiting App Tabs

One of the powerful features of a custom object is the fact that it can have relationships with other objects in the system. For example, for every review written by an interviewer and entered into the system, we'll want to associate it with the job application of the candidate who was being interviewed. Again, we won't need to write any SQL to make this happen—thanks to the platform, defining a relationship will be as simple as a few clicks of the mouse.

Security and Sharing Rules

Another important function that we'll need to build into our app is the ability to restrict access to data that particular users shouldn't see, without preventing other users from performing their jobs effectively. We're going to implement this requirement with a group of components that we've grouped under a single term: *security and sharing rules*.

With security and sharing rules, we'll first specify which custom objects a particular user should be allowed to create, view, or edit (for example, Candidate and Position), and then which instances of those objects should be accessible (for example, the records for candidate John Smith or the Senior Sales Manager position). Controlling our data either with the wide brush of object-level security or with the more detailed brush of record-level security will give us a lot of power and flexibility in controlling what users can and can't see.

Workflow and Approval Processes

Three of our requirements involve automating business processes, such as triggering an alert email to a recruiter whenever a job application's status has changed, and submitting new job

openings for executive approval. Once again, the Force.com platform makes these requirements easy for us to implement natively with the built-in *workflow* and *approval process* components.

Workflow and approval processes allow us to create business logic based on rules:

- Workflow rules can assign tasks to users, update fields, or send email alerts
- Approval processes allow users to submit sensitive records like new contracts or purchase orders to other users for approval

For example, in our Recruiting app, we can create a workflow rule that triggers an event whenever the status of a job application has changed to Reject or Extend an Offer, as illustrated below.

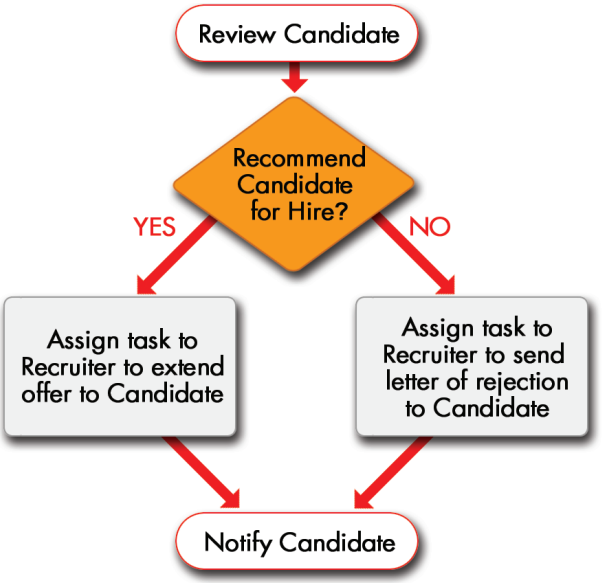


Figure 5: Workflow When a Job Application's Status Has Changed

When a hiring manager makes a decision to either extend an offer to or reject the candidate, changing the status of the application triggers the appropriate task to be assigned to the recruiter for that position. Based upon the hiring manager's decision, the recruiter performs the appropriate follow-up task.

Similarly, we can define an automatic approval process that sends all new positions to the appropriate management for approval. If the position is approved, its status automatically changes to Open - Approved and recruiters can start the hiring process. If the position is rejected, its status automatically changes to Closed - Not Approved and the position won't be filled.

Custom Reports and Dashboards

Finally, we need to give users a way to inspect the status of all positions and job applicants in the Universal Containers recruiting program. Managers need to delve into the intricate details of how each recruiter is performing, while executives just want a high-level overview of how departments are doing with staffing goals.

We can meet these requirements with the custom report wizard and dashboards. The wizard allows us to create detailed reports with filters, conditional highlighting, custom subtotals, and charts, while dashboards allow us to display a group of up to 20 different report charts on a single page.

Composite Components

Although we'll be able to use native platform functionality to satisfy most of our Recruiting app use cases, there's still one use case that won't be so easy to implement: the ability to map the locations of all candidates who are applying for a particular position.

At this point, we'll need to leave the relative comfort of the platform's native components and cover the gap by building a composite component, leveraging functionality from another website like Yahoo! Maps. Although we'll have to write a little code to make this work, the integrated component lives in a tab and looks just like any other part of our custom app.

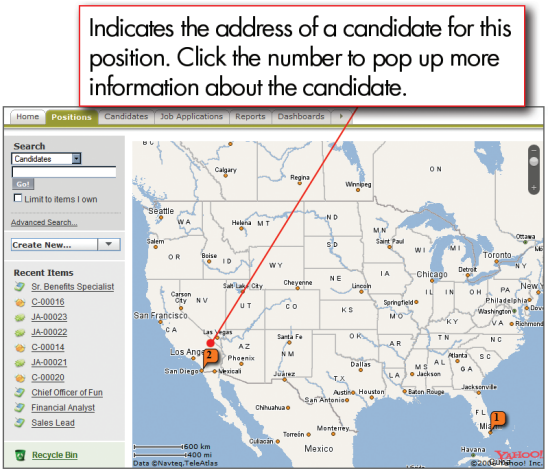


Figure 6: The Candidate Mapping Tool

Although we haven't yet gone into detail about how any of this stuff is going to work, you can probably see now just how flexible and powerful the Force.com platform can be when you're creating a custom app.

In the next chapter, we'll start out by building our first custom object. We'll swiftly get a feel for how the platform interface works, and it won't be any time at all before you're building app components easily and quickly. When it's this easy, you can't help but become an expert!

Chapter 3

Reviewing Database Concepts

In this chapter ...

Now that we've introduced the power of the Force.com platform and learned about the requirements of the Recruiting app that we're going to be building, let's take a moment to talk about databases and why a simple understanding of database concepts can help you realize the full potential of the platform and make your app development a whole lot easier.

As you know, the underlying architecture of the platform includes a database where your data is stored. This means that all of the information you enter is stored in that database and then retrieved from the database whenever you view it within your app.

Historically, companies were required to buy, build, and maintain their own databases and IT infrastructures in order to distribute and run their applications. On-demand computing on the Force.com platform provides an alternative and makes it easy for you, as a company or as a sole developer, to build and deliver your app. Part of the simplicity of the on-demand model is that the technical responsibilities of maintaining and running all of the database hardware and software is handled by the hosting company (in this case, salesforce.com), so you can focus on developing your app.

It's worth pointing out that although your data is stored in a database and a simple understanding of database concepts is helpful, you don't need to be a database developer to build an app on the platform. We won't be doing any traditional database programming in the course of developing our app.

What's a Database?

In simple terms, a *database* is an organized collection of information. Common examples include a phone book, a library catalog, an employee directory, a catalog of the MP3s you own, or in the case of our Recruiting app, information about the open positions at a company, the people who are applying for those positions, and the managers at our company who are in charge of hiring each position.

Typically, you use a database to collect information about people, things, or concepts that are important to you and whatever project you're working on. In standard database language, the category of person, thing, or concept you want to store information about is referred to as an *entity*, although in standard Force.com terminology, we refer to this as an *object*.

In a database, each entity is represented by a *table*. A database table is simply a list of information, presented with rows and columns, about the category of person, thing, or concept you want to track. So in a phone book, you might have a table to store information about residences and another table to store information about businesses; or in a library catalog, you might have one table to store information about books and another to store information about authors.

In our Recruiting app, we'll have one table to store information about open positions, another table to store information about the candidates applying for the positions, and a table to store information about hiring managers. (Our Recruiting app will have more than just this, but we'll get to that later.)

In very simplistic terms, a Force.com object is similar to a database table in that you'll have a separate object for each person, thing, or concept about which you want to collect information. In reality, a Force.com object is much more than this because the full functionality of the platform is behind each object. Each object automatically has built-in features like a user interface, a security and sharing model, workflow processes, and much more that you'll learn about in the rest of this book.

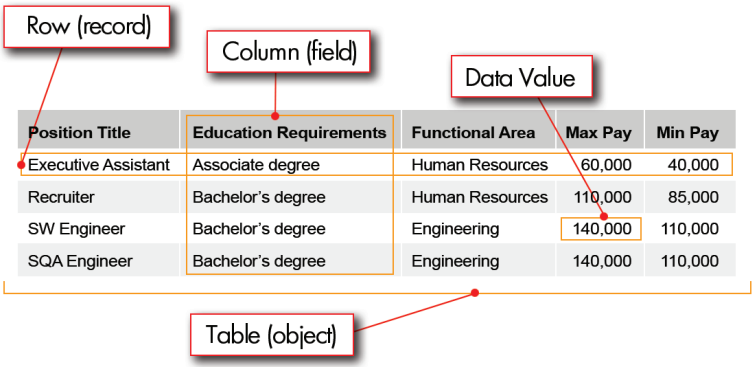
Note: As we introduce database concepts, "object" and "table" will be used interchangeably because they are similar. Just remember that a Force.com object is much more than just a database table.

It's important to understand that a single database table, or Force.com object, should contain only one type of information. You don't want to lump all of your information into one table, so you wouldn't store positions, candidates, and hiring managers all in the same place. Not only is this not good database design, but it doesn't allow you to relate objects to one another. For example, if all of our data were in one table, how would we ever know which candidates were applying for which positions, or which managers were in charge of hiring for which positions?

As we define our app, it's important for us to keep this in mind and ask ourselves questions like, "What kind of information do we want to store? Can we separate our information into distinct categories so that each object holds only one type of person, thing, or concept?" The answers to these questions will guide us as we design the structure of our application.

What's in a Database?

As we mentioned, a database table presents your information in rows and columns. Let's take a look at how a table of positions might look:



Position Title	Education Requirements	Functional Area	Max Pay	Min Pay
Executive Assistant	Associate degree	Human Resources	60,000	40,000
Recruiter	Bachelor's degree	Human Resources	110,000	85,000
SW Engineer	Bachelor's degree	Engineering	140,000	110,000
SQA Engineer	Bachelor's degree	Engineering	140,000	110,000

Figure 7: Position Information in a Table

Each row in the table represents the information about a specific instance of the object, for example, the Recruiter position or the SW Engineer position. In standard Force.com terminology, we call this a *record*. For every object you want to track in your app, you'll have multiple records to represent each individual item about which you're storing information. It's common for users who are new to the platform to confuse the meanings of object and record. It'll make your development a lot easier if you remember that an object is a category of information, such as a position or candidate, and the record is a single instance of an object, such as a SW Engineer.

Note: As a side note here, we'll mention that the platform includes a set of built-in objects when you first start using it; we call these *standard objects*. One example of a standard object is the User object, which stores information about each person who is a user of the app, like our hiring managers. You can also build your own objects to store information that's unique to your app; we call these *custom objects*. Both standard objects and custom objects are not really all that different—one kind is prebuilt for you, and the other you build yourself. We'll talk more about these later as you start to build your app.

Now let's look at the columns in the table. Each column lists a particular piece of information such as the Position Title or Max Pay. We refer to these as *fields*. Every object has a set of fields that you use to enter the information about a particular record. For each field in the table, a single item of data that you enter, such as the Functional Area of "Human Resources," is referred to as a *data value*.

Just like objects, fields come in two varieties: standard and custom. The standard fields are the ones that are built into the platform and automatically added for you. The custom fields are the ones you define to store specific pieces of information that are unique to your app. Fundamentally, there is no difference between standard and custom fields. Both are simply columns in the database table. We'll talk more about standard and custom fields later when you begin building your app.

What's a Relational Database?

Now you have some information stored in your database, but so what? You could easily make a list of positions using Microsoft Excel or some other spreadsheet software. For each position, you could even list the hiring manager in a field called Hiring Manager, like this:

Position Title	Education Requirements	Functional Area	Max Pay	Min Pay	Hiring Manager
Executive Assistant	Associate degree	Human Resources	60,000	40,000	Phil Katz
Recruiter	Bachelor's degree	Human Resources	110,000	85,000	Megan Smith
SW Engineer	Bachelor's degree	Engineering	140,000	110,000	Andrew Goldberg
SQA Engineer	Bachelor's degree	Engineering	140,000	110,000	Ben Stuart

Figure 8: Position Information with Hiring Manager Field

But what if a hiring manager is responsible for hiring more than one position? You would need to have duplicate records for the same hiring manager so you could capture every position for which that hiring manager is responsible, like this:

Position Title	Education Requirements	Functional Area	Max Pay	Min Pay	Hiring Manager
Executive Assistant	Associate degree	Human Resources	60,000	40,000	Phil Katz
Recruiter	Bachelor's degree	Human Resources	110,000	85,000	Megan Smith
Senior Recruiter	Bachelor's degree	Human Resources	115,000	95,000	Megan Smith
SW Engineer	Bachelor's degree	Engineering	140,000	110,000	Andrew Goldberg
Senior SW Engineer	Bachelor's degree	Engineering	140,000	110,000	Andrew Goldberg
SQA Engineer	Bachelor's degree	Engineering	140,000	110,000	Ben Stuart

Figure 9: Position Information with Duplicate Hiring Managers

This is not a good database design! Using this approach, data is repeated unnecessarily. In addition, there is really no way to capture additional information about our hiring managers, like their email addresses or phone numbers. And if we try to add information about which candidates are applying for each position, you can imagine that our simple table will quickly become extremely complex and unmanageable.

As we mentioned before, you want to create separate database tables, or objects, for each person, thing, or concept you want to track. A better way to model our scenario here would be to create one object for positions, one object for candidates, and one object for hiring managers. (Luckily, the platform has a standard object that we'll be able to use to represent our hiring managers—the User object.)

Once we have our data separated into discrete objects, we can easily relate objects to each other. This is what a relational database is all about! A *relationship* is an association between two or more tables. For example, we can relate positions to hiring managers so we know which positions each hiring manager is responsible for:

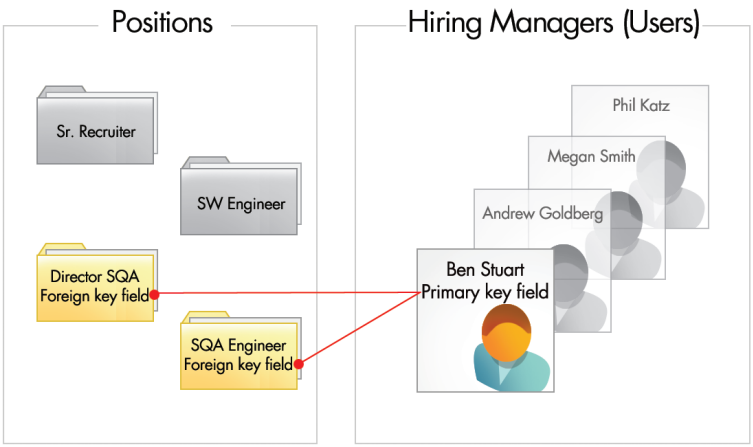


Figure 10: Positions Related to Hiring Managers

From a technical standpoint, 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 primary key is one part of what defines the relationship; the other part is the *foreign key*. A foreign key is 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. 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.

Primary and foreign keys are fundamental to the concept of relationships because they enable tables to be related to each other. As you begin building your app, you won't really need to think too much about primary keys and foreign keys. The important concept to understand here is that in a relational database, objects are related to each other through the use of common fields that define those relationships.

Summary of Database Concepts

At this point, we're ready to dive into the building of our Recruiting app. But first let's recap what we've learned about databases. Whether this was your first introduction to databases or whether you're already an experienced database developer who's new to the Force.com platform, the important things to remember are:

- A *database* is an organized collection of information.
- A database table stores information about a single type of person, thing, or concept—such as a job position. In the Force.com platform, we use the term *object* here (even though an object is much more than this, as you'll see).

- A database row, or *record* in Force.com terms, represents a single instance of an object—such as the SW Engineer position.
- A *field* stores a particular piece of information on a record.
- *Relationships* define the connection between two objects, and objects are related to each other through the use of common fields.

Now that we've got that all covered, let's get started building our first object!

Chapter 4

Building a Simple App

In this chapter ...

Just as traditional programming books first teach you how to write a simple "Hello World" program before getting into more complicated things, in this chapter, we're going to create a very simple version of the Recruiting app to show you just how easy it is to get started with the Force.com platform. Along the way we'll orient ourselves to the platform's user interface (where we'll be doing most of our work), and we'll learn how to create and configure our first custom object. Although easy and straightforward, the tasks we complete here will be the first step in developing a full-featured Recruiting app. So let's dive right in!

Becoming Familiar with the Setup Area

Since we're going to spend most of our time working in the Setup area of the platform, let's first become familiar with what it is and how to navigate to it.

The Setup area is an administrative tool and user preferences area, all in one. We perform almost every task we need to create our app in the Setup area, so most of the "Try It Out" sections of the book are going to start with an instruction like, "Click **Setup ➤ Build ➤ Custom Apps**." This is a short way of saying:

1. Click the **Setup** link in the top-right corner of the page (shown in the following screenshot).
2. Go to the App Setup area on the left side of the page.
3. Click the **+** icon to expand the **Build** menu, or just click the **Build** link.
4. Click the **Custom Apps** link.

The final link that you click (in this example, **Custom Apps**) will change depending on the task you're trying to perform, but you get the general idea.

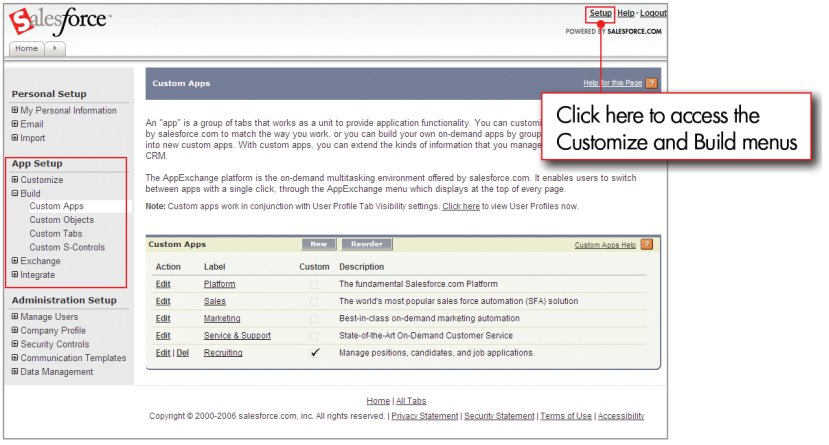


Figure 11: The Setup Area

Similar to the other parts of the application, the Setup area consists of a tab bar, a navigational sidebar, and a main window:

- The tab bar is made up of the same tabs that appear in the regular application. Just click on any one of the tabs to exit the Setup area and go to that tab in the main application.
- The navigational sidebar includes expandable lists of all the tools that are available in the Setup area:

Personal Setup
: These tools control individual preferences and are available to all users.

App Setup
: These tools configure the standard objects, custom objects, and custom apps that are deployed and are typically used only by administrators.

Administration Setup
: These tools configure the platform as a whole and are typically used only by administrators.

- The main window is where the navigational links or a selected setup tool are actually displayed.

Now that we know what we're looking at, let's start creating our simple app.

Introducing Apps

What should we do first? If we were writing a software application, the first thing we'd need to do is build a project where we could store all the code that we were going to write. With the Force.com platform, the first thing we need to do is create a new app.

Like a programming project, an *app* is little more than a container for all of the objects, tabs, and other functionality that we're going to build as part of our Recruiting application. It consists simply of a name, a logo, and an ordered set of tabs. The simplest app contains only one tab—the Home tab—and a default logo. As we define more tabs in the remainder of this book, we can add them to the app later.

Let's start clicking through the process of actually creating a simple app now. Log in to the Salesforce Developer Edition account that you signed up for earlier so you can follow along!

Note: Because the platform is continually evolving, you might find that the screenshots you see in this book vary slightly from what you see on your screen. These changes should be minor and shouldn't affect your understanding.

Try It Out: Defining an App

To create an app:

1. Log in to your Developer Edition account using your username and password. Typically, we refer to any Salesforce account as an "organization," so be aware of that as we proceed through the rest of the book.

2. Click **Setup ➤ Build ➤ Custom Apps**.
3. If you see an introductory splash page, simply click **Continue**.

Note: You'll find that many parts of the application have these splash pages to help you understand what you can do with the platform. If you never want to see a particular page again, just click **Don't show me this page again**.

Welcome to the Custom Apps list page! Like many of the setup tools, the starting page for the Custom Apps tool consists of a list of all the apps that are currently enabled for your organization. Depending on what you've already purchased from salesforce.com or downloaded from the AppExchange directory, you'll probably already have some standard apps listed here.

4. Click **New**.

   The New Custom App wizard appears.

5. In the `Label` field, enter Recruiting.

The label is the name that will represent our new app in the AppExchange app menu that appears at the top right of all pages. Users can use this menu to switch back and forth between apps.

Notice that a vertical red bar appears just to the left of this `Label` field. This red bar indicates that you must provide a value for this field in order to save your work. If you don't enter a value here and try to proceed, an error message is displayed, as shown in the following screenshot.

Figure 12: Required Fields Highlighted in Red

6. In the `Description` field, enter "Manage positions, candidates, and job applications."
7. Click **Next**.

The next screen in the New Custom App wizard allows you to specify the image file that should be used for this app's logo. Whenever the app is selected in the AppExchange app menu, this is the logo that appears in the upper-left corner of all pages. Since we're just creating a simple app, let's accept the default Salesforce logo that's already provided. We can always change it later.

8. Click **Next**.

As we said before, an app is a container for an ordered collection of tabs, and this step of the New Custom App wizard allows us to specify which tabs we want to include in our new app. The `Available Tabs` list shows us the standard and custom tabs that are available for us to choose, and the `Selected Tabs` list shows us which tabs are already included, listed in the order that they should be displayed. You'll notice that one tab, the Home tab, is already included in our app by default. This is because the Home tab is required in every app, and must always be in the first position.

Again, since we're just creating a simple app, let's accept the default and move on. We'll add more tabs later.

9. Click **Next**.

Now that we've defined the label, logo, and tabs that should make up our app, you might be wondering what remains to be done in the New Custom App wizard—shouldn't we already be done? It turns out that one crucial step remains: we need to define the users who will be allowed to access our app.

In this step of the New Custom App wizard, we can choose which user profiles should have access to the app. We'll learn more about profiles in *Chapter 7: Securing and Sharing Data* on page 109. For now, just understand that every user is assigned to a profile, and profiles control which apps the users assigned to that profile can view.

10. Select the `Visible` checkbox next to the Standard User and System Administrator profiles.
11. Click **Save**.

That's it!

Look at What We've Done

Now that we've made it back to the Custom Apps list page, let's see what we've just done. First of all, we've got a new entry in the Custom Apps list—our Recruiting app! It shows up

at the bottom of the list, signifying that it's going to show up at the bottom of our AppExchange app menu. In fact, let's go look at the AppExchange app menu now.

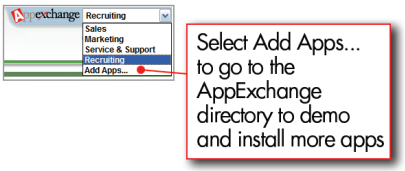


Figure 13: AppExchange App Menu

Tip: If you want to change the position of our app in this menu, do so from the Custom Apps list page by clicking **Reorder** and rearranging the available apps as you see fit.

Now select the Recruiting app from the menu and see what happens—our app is launched with a single Home tab! We've created the Recruiting app's Home tab, and we've added it to the AppExchange app menu. That's how easy it is to get started.

You'll notice that the approach we're taking here is iterative: we'll build part of the app, look at what we've accomplished, and then add to it. This sequence not only reflects the fact that we're leading you through the steps of building an app in this book, but you'll also find that in building Force.com apps in general, this iterative process is common.

During the course of this book, you'll also notice that unlike with traditional coding projects, your app is always functional. There's no build or compile phase, and as a result, you'll almost never be chasing down syntax bugs or other typos. In fact, with this simple one-tab app, you can already utilize all of the built-in functionality that comes with the platform, including search, calendar events and tasks, user preferences, and a familiar user interface.

Introducing Objects

Now that our app is functional (but rather boring), let's make it a little more interesting by introducing our first object.

As you might remember from the last chapter, an *object* is very similar to a database table in the Force.com platform. While the platform already comes with a number of standard objects that support default apps like Salesforce SFA and Salesforce Service & Support (for example, contacts, accounts, and cases), we can also define custom objects that allow us to store information specific to our Recruiting app.

Whether they're standard or custom, Force.com objects not only provide a structure for storing data but they also power the interface elements that allow users to interact with the data, such as tabs, the layout of fields on a page, and lists of related records. Because any object can correspond to a tab, and an ordered collection of tabs makes up an app, objects make up the heart of any app that we create with the platform.

With custom objects being so important—they have lots to do with how our app will look, behave, and feel—what we do with custom objects and how we use them quickly becomes essential to creating a successful app. The design of the data model behind an app is typically the biggest factor in its success or failure.

That's enough talk about objects for now. Let's go define one!

The Position Custom Object

The first custom object that we'll create for our Recruiting app reflects a typical recruiting task: describing a position. Recruiters at Universal Containers need to keep track of all the positions they're hiring for, such as a Senior Developer, Sales Engineer, or Benefits Specialist. They'll need easy access to all positions in the system through a tab, and they'll need to include certain information for each position, such as its minimum and maximum salary range, the position's location, and its hiring manager. In Force.com terms, we'll create a custom object, create a custom tab for that object, and then define some custom fields.

Try It Out: Defining the Position Custom Object

To create the Position object, we're going to go back to the Setup area.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. On the All Custom Objects page, click **New Custom Object**.

Unlike defining a custom app, which we did through the New Custom App wizard, defining a custom object is confined to just one page. You'll find that the platform uses wizards or single pages depending on the amount of information that needs to be specified.

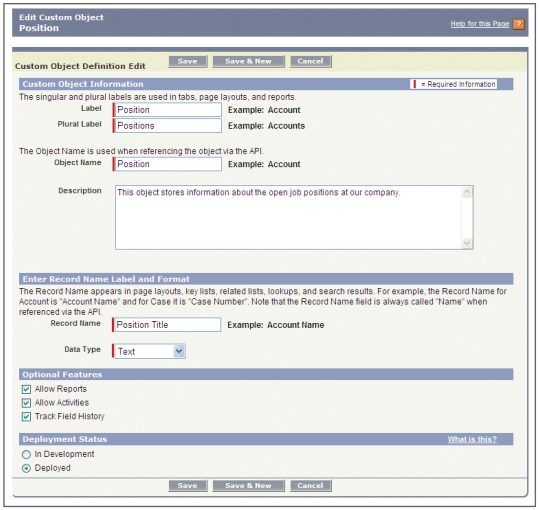


Figure 14: Custom Object Definition Page

3. In the `Label` field, enter Position.
4. In the `Plural Label` field, enter Positions.
5. The `Object Name` field is defaulted to Position. Let's leave it as is.

The `Label` and `Plural Label` of a custom object are what users see in all of the object's related user interface elements, such as the object's tab or in search results headings. Object labels work best as nouns, and the plural label is always used to label a custom object's tab (if you create a tab for your object).

The value of a custom object's `Object Name` represents the unique name for the object when it's referenced in the Force.com Web Services API. This value is helpfully autogenerated based on the value that you enter for the `Label`, except that all spaces and punctuation are replaced with underscore characters. We'll talk more about the API in *Chapter 10: Moving Beyond Native Apps* on page 225. For now, just keep in mind that the `Object Name` value must be unique across all objects defined in your organization.

Note: Within the platform, `Object Name` is actually stored with `__c` appended to the end as a suffix (for example, `Position__c`). This identifies it as a custom object.

6. In the `Description` field, enter "This object stores information about the open job positions at our company."
7. For the `Context-Sensitive Help Setting`, accept the default. If you later want to provide customized help documentation for your users about this object, you can come back and choose the `Open a window using a custom s-control` option.

8. In the `Record Name` field, enter Position Title.

The `Record Name` is the label for the field that identifies individual Position records in the system. A custom object cannot be saved without this identifying field.

9. In the `Data Type` drop-down list, select Text.

The `Data Type` drop-down list allows you to select the type of value that should be used for this identifying field: either Text or Auto-Number. Some objects, like Positions or Accounts, can be identified with a text field because there will always be a name for a position or account available. Other objects, like a Case (used in the standard Service & Support app) are harder to identify with a single text field, so we would assign them auto-numbers instead.

Tip: Whenever possible, it's best to use text as the data type for an identifying field so that users can more easily identify a particular record when several of them appear together in a single list.

To illustrate how custom object and record name labels work together in the app, let's fast forward a bit to see where each label will appear once we've defined our Position custom object, its tab, and a single Sr. Developer position record.

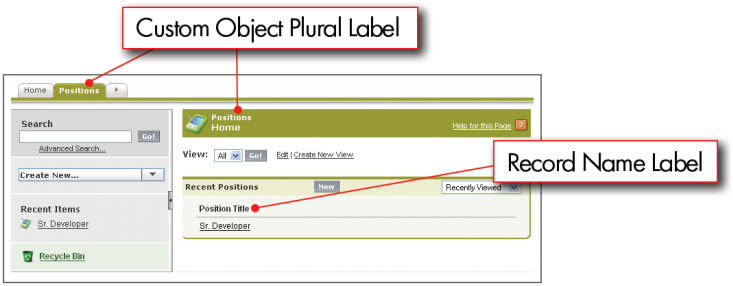


Figure 15: Custom Object and Record Name Labels

Let's move on.

10. In the Optional Features area, select the `Allow Reports`, `Allow Activities`, and `Track Field History` checkboxes.

These three checkboxes actually enable some really robust functionality:

Allow Reports

Selecting this option makes the data in the Position records available for reporting purposes. The platform comes with a large number of standard reports, and users can also create custom reports by using a simple yet

powerful reporting wizard. (To find out more about reports, see *Chapter 9: Analyzing Data with Reports and Dashboards* on page 193.)

Allow Activities

Selecting this option allows users to associate tasks and scheduled calendar events with a particular position. For example, a user could create a task, such as "Update salary range for Sr. Developer position," and specify attributes such as priority, due date, and status. The user could then handle the task herself or assign it to someone else. (To find out more about tasks, see "What is an Activity?" in the online help.)

Track Field History

Selecting this option allows the platform to automatically track edits to Position records, such as who changed the value of a field, when it was changed, and what the value of the field was before and after the edit. History data is available for reporting, so users can easily create audit trail reports when this feature is enabled. (To find out how to select which data is tracked, see "Tracking Field History" in the online help.)

In general, you should select these options if there's any chance that they might be useful for whatever custom object you're defining.

11. In the Deployment Status area, select `Deployed`.

Note: This step assumes that you're working in a development environment. If you're not, and if you don't want users to see the Position object after you click **Save**, select `In Development`. Setting the status to `In Development` hides Position records from all users except those with the "Customize Application" user permission (that is, just about anyone who isn't a System Administrator).

12. In the Object Creation Options area, select the `Add Notes & Attachments related list to default page layout` and `Launch New Custom Tab Wizard after saving this custom object` checkboxes.

These two options are available only when you're creating a new custom object. If you later decide to go back and edit some of the details about your custom object, you won't see them. But what do they do?

- Enabling notes and attachments for an object means that you can attach external documents to any Position record, in much the same way that you can add a PDF or photo as an attachment to an email. It's handy functionality, so you should generally select it.

- Launching the New Custom Tab wizard does exactly what it says—it's a shortcut to launching the tab wizard after we've saved our Position object and will save us a few clicks if we know that we need a tab.

All set? Let's go ahead and save our Position custom object now.

> **13.** Click **Save**.

That's all there is to it! As promised, the New Position Tab wizard is displayed instead of the list of custom objects that we'd normally see. Let's take a moment to talk about why we should even be defining a tab for our Position object in the first place. What's so great about tabs, anyway?

Introducing Tabs

If you're familiar with the Force.com platform, you know that clicking tabs is how you navigate around an app. Every 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 home page for that object appears. For example, if you click the Accounts tab, the Accounts tab home page appears, giving you access to all of the account records that are defined in your organization. Click the name of a particular account record and you'll view all of the record's information in its associated detail page.

What's really powerful about building an app with the platform is that you can create custom tabs that look and behave just like the tabs for standard objects that are already provided. From the perspective of your end users, any customizations that you make appear perfectly seamless, and as a developer, you don't have to do anything special to make it work that way! Let's see how quickly we can create a tab for our Position object.

Try It Out: Defining the Positions Tab

To create a custom tab for our Position object, we're going to use the New Custom Object Tab wizard that was so helpfully launched for us when we clicked **Save** after defining the object. However, in case you forgot to select the `Launch New Custom Tab Wizard after saving this custom object` option or if you're coming back to work that you previously saved, have no fear! There's another way to launch the wizard.

> **1.** Click **Setup ➤ Build ➤ Custom Tabs**.
> **2.** In the Custom Object tabs area, click **New**.

Easy. Now that we're all on the same page, let's get started working through the wizard.

3. In the `Object` drop-down list, select Position.

If you launched the wizard directly after defining the custom object, the Position object is automatically selected for you.

4. Click the `Tab Style` lookup icon to launch the Tab Style Selector as shown in the following screenshot.

Every object that appears as a tab must have a unique color scheme and icon. This color scheme is what identifies the object, not only on its tab but also in different places in the user interface, such as in related lists and search results.

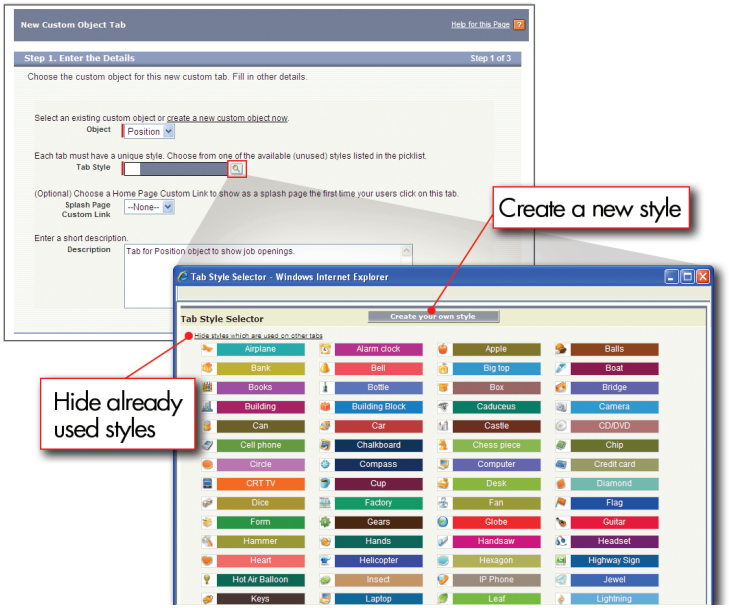


Figure 16: Custom Object Tab Setup Page and Tab Style Selector

In the Tab Style Selector, you can choose a predefined color and icon or you can create your own. To keep things simple, we're going to select an existing style.

5. Click the **Hide values which are used on other tabs** link to make sure you choose a unique style.
6. Click any colored box to choose a color scheme and icon.

Leave the `Splash Page Custom Link` drop-down list set to --None--. We'll learn more about custom links in *Chapter 10: Moving Beyond Native Apps* on page 225.

7. In the `Description` field, enter "A tab and color scheme for the Position custom object."
8. Click **Next**.
9. Click **Next** again to accept the default user profile visibility.

Just as we controlled access to our Recruiting app by selecting user profiles in the New Custom App wizard, we can also control access to our Positions tab by selecting user profiles here. We'll learn more about user profiles and what they do in *Chapter 7: Securing and Sharing Data* on page 109. For now, just know that accepting the defaults will make the tab visible to all users.

10. Deselect all of the `Include Tab` checkboxes except the one for our Recruiting app.

In performing this step, we're providing access to the Positions tab only when someone has access to our Recruiting app. Unless an employee is interested in recruiting, he or she probably doesn't need to see this tab.

11. Select the `Append tab to users' existing personal customizations` checkbox.

If you don't select this option, any users who have personalized their tab display will not immediately see the Positions tab. Also, if you've already created a new tab and didn't turn this option on, you have to first delete the existing tab and then re-create it with this option turned on to automatically push the tab to existing users. What a pain! Do yourself a favor and just always keep this option selected.

12. Click **Save**.

You'll notice when the page refreshes that the Positions tab has automatically been added next to the Home tab at the top of the page.

Look at What We've Done

To truly appreciate what we've just built with a few clicks, let's take a look at what we've done.

1. First, click the Positions tab to display the Positions tab home page, as shown in the following screenshot. Although the list is empty because we haven't yet created any records, you can see how this page will become the gateway to viewing, creating, editing, and deleting all of the positions that we create in our Recruiting app. It looks just like the tab home page of any other standard object.

Figure 17: The Positions Tab Home Page

2. Now, check out the contents of the **Create New...** drop-down list in the left sidebar. As promised, our custom object has been seamlessly incorporated into the platform with the other standard objects like Event and Task. An end user need never know that the Positions tab was created with a custom object, because it shows up alongside the standard objects as well.
3. Select Position from the **Create New...** drop-down list, or click **New** in the Positions tab home page. Voilà—it's the Position Edit page! Sadly, though, our position still doesn't have much room for data. At this point, all we have is a field for `Position Title` (the record identifier) and `Owner`, a default field that appears on every object to identify the user who created the object.
4. Click **Cancel**. It doesn't do to create a Position record with hardly any interesting data. We need more fields! And sure enough, that's what we'll get to next. First, though, let's revisit our Position custom object and orient ourselves to what else is available through a custom object detail page in the Setup area.

Becoming Familiar with Setup Detail Pages and Related Lists

You may recall when we first introduced the concept of objects that we learned: "Whether they're standard or custom, Force.com objects not only provide a structure for storing data but they also power the interface elements that allow users to interact with the data, such as tabs, the layout of fields on a page, and lists of related records." If you've been following along closely, you might have been wondering why we didn't get to define any fields (other than the identifier field of Position Title) or user interface elements (other than the Positions tab) when we created our Position object. If fields and user interface elements are a part of the definition of what a custom object is all about, where do we get to define them?

It turns out that the Force.com platform differentiates between the initial creation of certain components and details related to those components. In other words, the information that we see when we define or edit a custom object is different from the information that we see when

we view a custom object that's already defined. Let's go back to our custom object list page to see how this difference is reflected in the platform interface:

1. Click **Setup ➤ Build ➤ Custom Objects**.

Here we are back in the custom object list page. You'll notice in the row for Position there are three links that we can click:

Edit

This link takes us back to the Custom Object Edit page where we originally defined our Position object.

Del

This link deletes the custom object, including any records, tabs, reports, or other components associated with that object.

Position

This link takes us to the custom object detail page for our Position object.

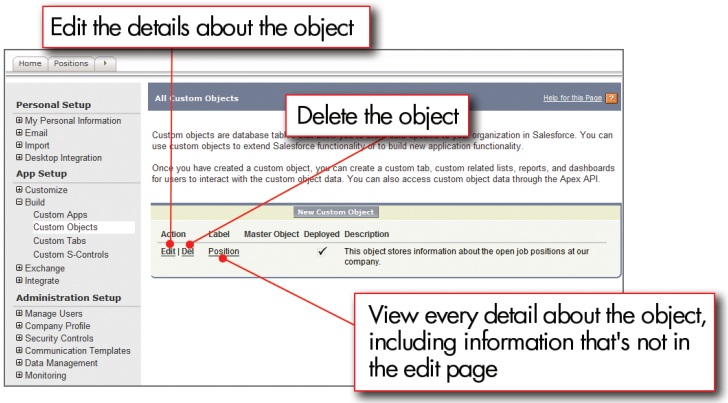


Figure 18: Custom Object List Page: Edit, Delete, and Detail Links

We're already familiar with the edit page from when we defined our Position object, and we certainly don't want to delete our object. Let's go ahead and open up the detail page to see what we can do there.

2. Click **Position**.

As you can see, the Custom Object Edit page that we filled out when we defined our Position object was just the tip of the iceberg. The top two areas of the Position detail page (see *Figure 19: Position Custom Object Detail Page* on page 47) include all of the information that we originally specified, plus a few standard fields that the platform includes with every object.

Below those areas are several additional groupings of data that allow us to do more with our Position object.

In Force.com terms, those groupings of data are called *related lists*, and they're a big part of what makes the platform so powerful. A related list is a list of records or other components that are associated with whatever we're viewing. Related lists appear in both the main application and in the Setup areas and represent a relationship between the items that appear in the related list and the object or record that we're viewing in the detail area. We'll learn a lot more about relationships in *Chapter 6: Expanding the Simple App Using Relationships* on page 81, but for now, just understand that anything that appears in an object's related list is directly related to that object.

Now that we've found out where we can continue customizing our Position custom object, let's use the Custom Fields & Relationships related lists to create some more fields in our Position object.

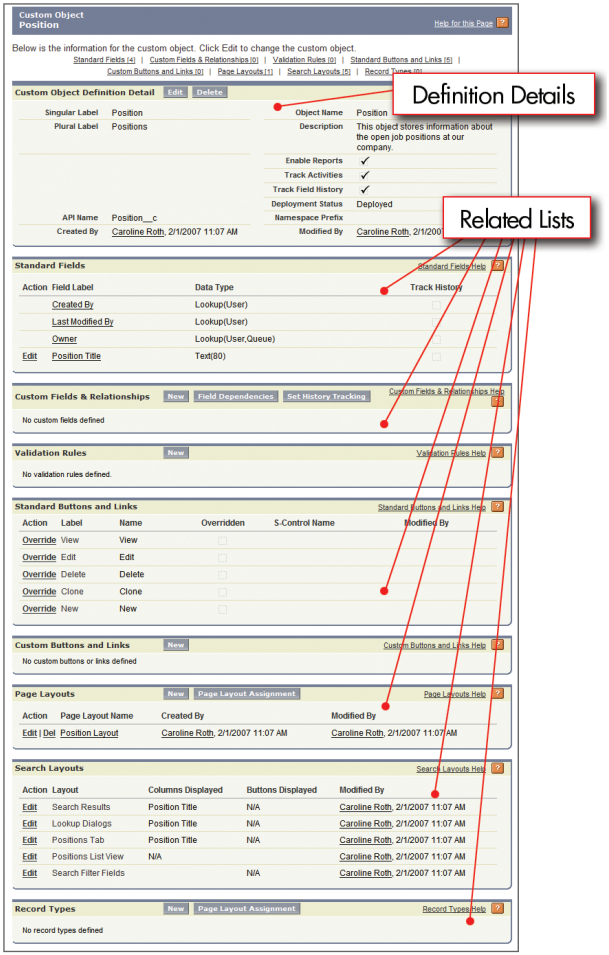


Figure 19: Position Custom Object Detail Page

Introducing Fields

We're ready to add more fields to our Position custom object, but first, let's talk briefly about what a field is and how it fits into the world of the Force.com platform.

As you might remember from the last chapter, a *field* is like a database column. The primary characteristic of a field is its data type—some fields hold text values, while others hold currency values, percentages, phone numbers, email addresses, or dates. Some fields look like checkboxes, while still others are drop-down lists or record lookups from which a user makes a selection.

The data type of a field controls the way the field is ultimately displayed in the user interface and how data entered into the field is stored in the platform. To get a better feel for how the fields will look, let's take a sneak peak at what the Position object is ultimately going to look like and the types of custom fields we're going to create for it:

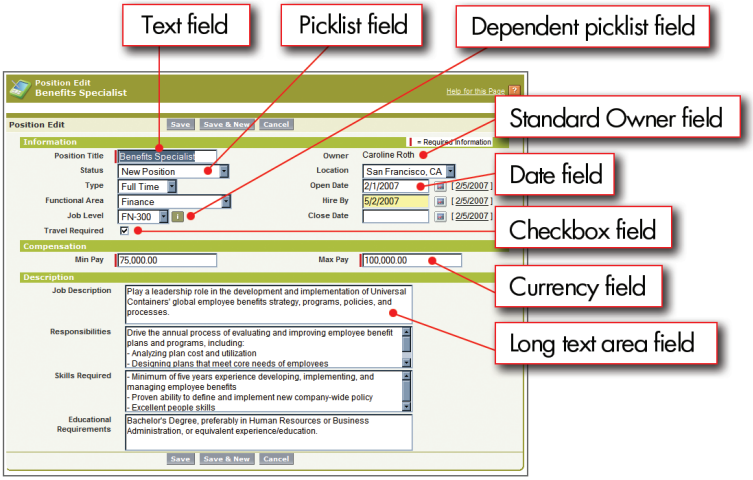


Figure 20: Position Custom Object Fields

There are a lot of fields here that we need to define, some more complicated than others. To keep things simple, let's go through and create the simple text, currency, checkbox, and date fields. We can tackle the more complicated picklist and custom formula fields in *Chapter 5: Enhancing the Simple App with Advanced Fields and Page Layouts* on page 55.

Try It Out: Adding Text Fields

First let's define a few text fields. We already created a basic text field for `Position Title` when we defined our Position custom object. Looking at our screenshot, the only text fields that remain are the text fields under the Description heading. We'll start by defining the Job Description field.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **New**.

Every time you create a custom field, you'll first choose a data type from the field type selection page.

The platform allows us to choose between different types of text fields:

- Basic text fields allow users to enter any combination of letters and numbers on a single line, up to as many as 255 characters.
- Text area fields also have a 255 character limit but also allow carriage returns so the text can be formatted on separate lines.
- Long text fields allow as many as 32,000 characters, on separate lines.

Since job descriptions can be lengthy, let's choose a long text area.

4. Choose the `Text Area (Long)` data type and click **Next**.

Tip: Carefully consider the data type you choose for each of your custom fields, because once you set it, it isn't always the best idea to change it later. See "Notes on Changing Custom Field Types" in the online help for details.

The second page of the custom field wizard allows us to enter details about our long text area field. The fields that appear in this step change depending on the data type that we selected in the previous page.

5. In the `Field Label` field, enter Job Description.

Like the other labels that we've seen in the platform so far, the `Field Label` specifies the text that should be displayed to users when the field is rendered in the user interface. Notice that when we enter a value for `Field Label`, `Field Name` is automatically populated with the same text but with all spaces and punctuation replaced by underscores. The value for `Field Name` is a unique name that is used to refer to the field when writing a custom formula or using the API.

Note: Within the platform, `Field Name` is actually stored with `__c` appended to the end as a suffix (for example, `Job_Description__c`). This identifies it as a custom field.

6. In the `Length` field, enter 32,000.

The `Length` field allows us to restrict the maximum number of characters that are allowed. Since we don't get any benefit from this kind of restriction, leave this value set to 32,000.

7. In the `# Visible Lines` field, enter 3.

This field allows us to specify how large our text box will appear on the page.

8. In the `Help Text` field, enter "High-level description of the job and what its duties are."

This help text is displayed on record detail and edit pages when users hover over the field's label. Its purpose is to assist users in filling out the field correctly. It's optional to add help text for a field, but it's a good idea if you have fields that you think users might be confused by.

There's no obvious default value for a text field, so just leave `Default Value` blank.

9. Click **Next**.

The third page of the Custom Field wizard allows us to restrict access to this field from certain user profiles. We'll learn more about profiles and field-level security in *Chapter 7: Securing and Sharing Data* on page 109, so for now, just accept the defaults.

10. Click **Next**.

The last page of the wizard allows us to automatically place our field on the Position page layout. Again, we'll learn about page layouts in the next chapter, so for now, just accept the defaults.

11. Click **Save & New**.

Instead of clicking **Save** and returning to the Position object detail page, clicking **Save & New** saves a few clicks and allows us to finish up the other text area fields that we need. Here's what you need to know to define them:

Table 2: Position Object Long Text Area Fields

Field Type	Field Label	Length	# Visible Lines	Default Value
Text Area (Long)	Responsibilities	32,000	3	Leave unspecified
Text Area (Long)	Skills Required	32,000	3	Leave unspecified
Text Area (Long)	Educational Requirements	32,000	3	Leave unspecified

Now that we've wet our feet with text fields, let's quickly create a few more fields of other types. You'll find that with few exceptions, they're all very similar to one another.

Try It Out: Adding Currency Fields

To keep track of a position's salary range, we need to add two currency fields: `Min Pay` and `Max Pay`. Note that unlike some fields, once we define these as currency fields, we won't be able to change them to any other type.

Defining a currency field is almost identical to defining a text field, with a few slight differences:

- The `Length` of a currency field actually corresponds to the number of digits to the left of the decimal point. An additional `Decimal Places` field handles the number of digits that should be displayed to the right.
- In the Details page of the wizard, a new checkbox called `Required` is displayed. We can select this option if we want to force our users to enter a value for this field when creating a new position.

Everything else should be familiar to you, so go ahead and use the custom field wizard to define the following fields:

Table 3: Position Object Currency Fields

Field Type	Field Label	Length	Decimal Places	Required	Default Value
Currency	Min Pay	7	2	Leave unchecked	Leave unspecified
Currency	Max Pay	7	2	Leave unchecked	Leave unspecified

Try It Out: Adding a Checkbox Field

Here's an easy one. The Position object requires a checkbox field to indicate if travel is required for the position. By default, this value should be unchecked. (Note that similar to currency fields, once you define a field as a checkbox, you can't change it to any other type.)

Use the custom field wizard one more time to define this field:

Table 4: Position Object Checkbox Field

Field Type	Field Label	Default Value
Checkbox	Travel Required	Unchecked

Try It Out: Adding Date Fields

Finally, before we close out this chapter, let's add three date fields to our Recruiting app to track the date a position opens, the date it closes, and the date by which it should be filled.

Date fields are great because they automatically include a popup calendar interface from which users can select a day without any typing—yet another built-in feature that we can leverage in our app without any extra work!

Once again we'll use the custom field wizard to define these three fields:

Table 5: Position Object Date Fields

Field Type	Field Label	Required	Default Value
Date	Open Date	Unchecked	Leave unspecified
Date	Hire By	Unchecked	Leave unspecified
Date	Close Date	Unchecked	Leave unspecified

Look at What We've Done

We've defined text, currency, checkbox, and date fields for our Position object. Let's take a look by going to the Positions tab and clicking **New**.

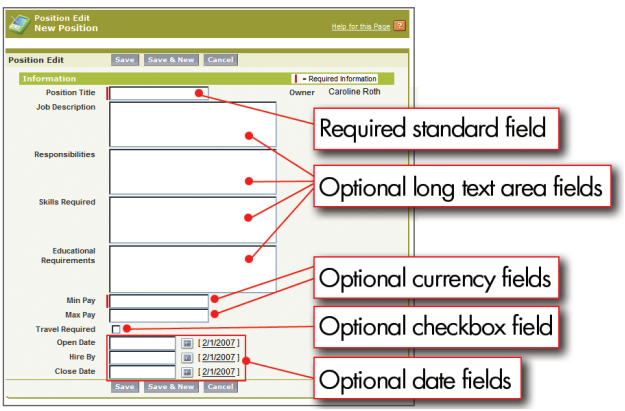


Figure 21: Position Object Fields

Check out all the fields we've just made! It's not the most elegant layout for all of our fields (each field got added to the page in the order that we created it), but it's definitely functional, and it looks just like any other page. Wasn't that easy?

Once again, welcome to the power of the Force.com platform. First we created a new recruiting app with a single Home tab, then we created a Position object and tab, and now we've just added a few fields, all in less than 15 minutes of clicking around. From start to finish we always

had a fully functional app, and we never had to spend any time compiling or debugging our "code"!

In the next chapter, we'll enhance our simple Recruiting app even further by adding some additional fields that are more complex, defining validation rules to help our data stay clean, and then moving the fields around in a page layout so users can more easily find and enter the information they need. Let's keep going!

Chapter 5

Enhancing the Simple App with Advanced Fields and Page Layouts

In this chapter ...

In the last chapter, we got our Recruiting app off to a quick start by defining the Position custom object, tab, and several simple fields. This simple version of our app had the same look and feel as any other page in the Force.com platform, and we were able to whip it together in a matter of minutes.

In this chapter, we're going to enhance the Positions tab: first by defining a few more advanced fields, then by defining a validation rule to make sure our data stays clean, and finally by moving our fields around within a page layout. These additions will help change the detail page of our Positions tab from a somewhat flat and inelegant user interface to something that users find powerful and intuitive to use. Let's get started!

Adding Advanced Fields

In this section, let's revisit the custom field wizard to help us create fields with more sophisticated functionality: picklists, dependent picklists, and fields that leverage custom formulas. We'll see how the platform's user interface helps guide us through the setup of these more complicated fields.

Introducing Picklists

When viewing the preview of what we wanted our Positions page to ultimately look like, there were several fields that were specified with drop-down lists. In Force.com terms, these fields are called *picklists*, and they consist of several predefined options from which a user can select.

Picklists come in two flavors: a standard picklist, in which a user can select only one option, and a multi-select picklist, in which a user can select multiple options at a time. For the purposes of our Position object, we need to define standard picklists for a position's location, status, type of job, functional area, and job level.

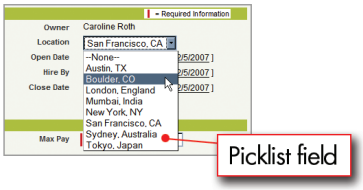


Figure 22: Location Picklist Field

Try It Out: Adding Picklists

Let's walk through the creation of the `Location` picklist field. Then, as in the previous chapter, we'll give you the information that you need to create the others on your own.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Picklist` data type and click **Next**.
5. In the `Field Label` text box, enter Location.
6. In the large text area box just below, enter the following picklist values, each on its own line:

    - San Francisco, CA

- Austin, TX
- Boulder, CO
- London, England
- New York, NY
- Mumbai, India
- Sydney, Australia
- Tokyo, Japan

7. Select the `Use first value as default value` checkbox.

This option allows us to populate the field with a default value. If you leave it deselected, the field defaults to None on all new Position records. Otherwise the field defaults to the first value that you specify in the list of possible picklist values. Because most positions at Universal Containers are based at its headquarters in San Francisco, CA, this should be the default.

8. Accept all other default settings for field-level security and page layouts.
9. Click **Save & New**.

Easy! Now specify the remaining picklists according to the table below:

Table 6: Status, Type, Functional Area, and Job Level Picklist Values

Data Type	Field Label	Picklist Values	Sort Alphabetically?	Use First Value as Default?
Picklist	Status	• New Position • Pending Approval • Open - Approved • Closed - Filled • Closed - Not Approved • Closed - Canceled	No	Yes
Picklist	Type	• Full Time • Part Time • Internship • Contractor	No	No

Data Type	Field Label	Picklist Values	Sort Alphabetically?	Use First Value as Default?
Picklist	Functional Area	• Finance • Human Resources • Information Technology • Retail Operations • Warehousing • Miscellaneous	Yes	No
Picklist	Job Level	• FN-100 • FN-200 • FN-300 • FN-400 • HR-100 • HR-200 • HR-300 • HR-400 • IT-100 • IT-200 • IT-300 • IT-400 • RO-100 • RO-200 • RO-300 • RO-400 • WH-100 • WH-200 • WH-300 • WH-400 • MC-100 • MC-200 • MC-300 • MC-400	Yes	No

Introducing Field Dependencies

Now that we've made all those picklists, answer this question: How many times have you clicked on a drop-down list and found far too many values to choose from? For example, maybe you were selecting Uruguay from a list of countries, and every country in the world was on the list. That meant that you had to scroll all the way down to the countries that started with the letter U. What a pain!

Fortunately, the folks who built the Force.com platform have encountered that situation a few times themselves, and as a result, they've given us a tool to help us avoid this problem with our own picklist fields: *field dependencies*.

Field dependencies are filters that allow us to change the contents of a picklist based on the value of another field. For example, rather than displaying every value for Country in a single picklist, we can limit the values that are displayed based on a value for another field, like Continent. That way our users can find the appropriate country more quickly and easily.

Picklist fields can be either *controlling* or *dependent* fields. A controlling field controls the available values in one or more corresponding dependent fields. A dependent field displays values based on the value selected in its corresponding controlling field. In the previous example, the Continent picklist is the controlling field, while the Country picklist is the dependent field.

Try It Out: Creating a Dependent Picklist

Looking at the picklists that we've created, it's quickly obvious that our users might get frustrated with the length of our `Job Level` picklist. Let's make our users happy by turning `Job Level` into a dependent field of the `Functional Area` picklist. Doing this will allow users to see only the four relevant job level values when a department is selected in the `Functional Area` picklist:

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **Field Dependencies**.
4. Click **New**.
5. For the `Controlling Field` drop-down list, choose Functional Area.
6. For the `Dependent Field` drop-down list, choose Job Level.
7. Click **Continue**.

A field dependency matrix displays with all the values in the controlling field across the top header row and the dependent field values listed in the columns below. For each possible value of the controlling field, we need to include the values that should be displayed in the dependent

picklist when that controlling value is selected. In the field dependency matrix, yellow highlighting shows which dependent field values are included in the picklist for a particular controlling field value.

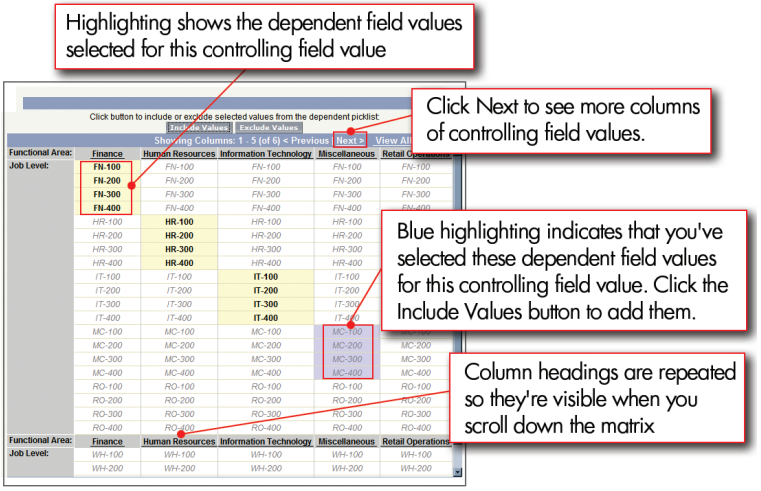


Figure 23: Field Dependency Matrix

To include a dependent field value, you simply double-click it. To exclude a dependent value from the list, double-click it again.

For example, let's try it out by including the values that should be displayed in the `Job Level` picklist whenever Finance is selected in the `Functional Area` picklist:

8. In the column labeled Finance, double-click FN-100, FN-200, FN-300, and FN-400.

Those four fields should now be shaded yellow in the `Finance` column.

Instead of double-clicking every `Job Level` value, we can also use Shift+click to select a range of values or Ctrl+click to select multiple values at once. Once those values are highlighted in blue, we can click **Include Values** to include them, or **Exclude Values** to remove them. Let's try it out.

9. In the column labeled Human Resources, single-click HR-100 and then press and hold the Shift key while clicking HR-400.
10. Click **Include Values**.

Now we have values selected for both the Finance and Human Resources columns!

11. Continue highlighting the appropriate values for all of the remaining columns, as described in the following table.

Tip: To get to all of the values that you need to modify for this step, you'll need to click **Previous** or **Next** to see additional columns.

Table 7: Functional Area and Job Level Field Dependency Matrix

Functional Area (Controlling picklist field)	**Job Level (Dependent picklist field)**
Finance	• FN-100 • FN-200 • FN-300 • FN-400
Human Resources	• HR-100 • HR-200 • HR-300 • HR-400
Information Technology	• IT-100 • IT-200 • IT-300 • IT-400
Retail Operations	• RO-100 • RO-200 • RO-300 • RO-400
Warehousing	• WH-100 • WH-200 • WH-300 • WH-400

Functional Area (Controlling picklist field)	Job Level (Dependent picklist field)
Miscellaneous	• MC-100 • MC-200 • MC-300 • MC-400

12. Click **Preview** to test the results in a small popup window.
13. Click **Save**.

Look at What We've Done

Now that we've created all those picklists, let's revisit the Positions tab to see what we have so far.

1. Go to the Positions tab.
2. Click **New**.
3. In the `Functional Area` picklist, select Finance.
4. Open the `Job Level` picklist.

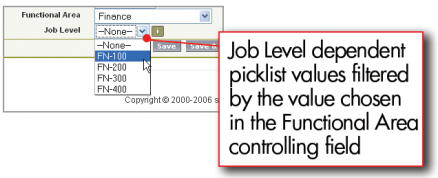


Figure 24: Dependent Picklist Fields

Our Recruiting app users are going to be very happy that they no longer have to deal with a long, onerous picklist. Now let's go add a field that's even more powerful and complex than a dependent picklist: a custom formula field.

Introducing Custom Formula Fields

Up to this point, the fields that we've defined have all had one thing in common—they each require a user to give them a value. Fields like that are very helpful for storing and retrieving

data, but wouldn't it be great if we could somehow define a "smart" field? That is, what if we could define a field that looked at information that was already entered into the system and then told us something new about it?

Fortunately, *custom formula fields* give us the ability to do just that. Just as you can use a spreadsheet program like Microsoft Excel to define calculations and metrics specific to your business, we can use custom formula fields to define calculations and metrics that are specific to our Recruiting app.

For example, on our Position object, we've already created fields for minimum pay and maximum pay. If Universal Containers gives out yearly bonuses based on salary, we could create a custom formula field that automatically calculated the average bonus that someone hired to that position might receive.

How would we perform this calculation if we were using a spreadsheet? The columns in our spreadsheet would represent the fields that we defined on our Position object, and each row of the spreadsheet would represent a different position record. To create a calculation, we'd enter a formula in a new column that averages the values of `Min Pay` and `Max Pay` in a single row and then multiplies it by a standard bonus percentage. We could then determine the average bonus for every position record row in our spreadsheet.

Custom formulas work in a very similar way. Think of a custom formula like a spreadsheet formula that can reference other values in the same data record, perform calculations on them, and return a result. However, instead of using cell references, you use *merge field* references. And, instead of typing characters in a box, you have a wizard to help you select fields, operators, and functions.

The net result is that anyone can quickly and easily learn to create formula fields. And, as with all platform tools, the on-demand delivery model makes it easy to experiment. You can create a formula, view the results, and change the formula again and again, as many times as you want! Your underlying data is never affected.

Tip: When defining your own custom formula fields, leverage the work of others. You can find more than a hundred sample formulas on the Successforce website at http://blogs.salesforce.com/features/2006/03/custom_formula_.html.

Calculating How Long a Position Has Been Open

Let's now think about another custom formula field that we could create for our Position object—a custom formula field that calculates how many days a position has been open. To do this, let's first think about the logic that we should use to define the field, and then we can go through the process of creating it in our Recruiting app.

Let's think about the data that we need to make this calculation: we need to know the current date and the date that the position was created. If we could somehow subtract these two, we'd have the number of days that the position has been open. Fortunately, it's easy to get both of these values:

- For the current date, we can simply use the platform's built-in `TODAY()` function. `TODAY()` returns today's date.
- For the date that the position was opened, we can use the `Open Date` field that we defined in the last chapter. Because we're using it in a custom field, we'll refer to it by its internal name, `Open_Date__c`.

Now that we have our two dates, we want to subtract them: `TODAY() - Open_Date__c`. Even if the two dates span different months or years, the platform is sophisticated enough to know how to handle all the intricacies of such a calculation behind the scenes. We just have to provide the dates, and the platform can do all the rest.

So far so good, but one problem still remains—what if the position has already closed? Our formula only works if we assume the position is still open. Once it closes, however, the value of our current formula will keep incrementing every day as `TODAY()` gets farther and farther away from the original `Open Date`. If we can, we want to use the `Close Date` field in the formula instead of `TODAY()` after a position closes. How can we do this?

Once again, all we need to do is dip into the extensive library of platform functions. The `IF()` function allows us to perform a test and then return different values depending on whether the result of the test is true or false. The `IF()` function's syntax looks like this:

```
IF(logical_test,
          value_if_true,
          value_if_false)
```

For the *`logical_test`* portion, we'll test whether the `Close Date` field has a value—if it does, the position obviously must be closed. We'll test for this with a third built-in function: `ISNULL()`. `ISNULL()` takes a single field and returns true if it contains a value and false otherwise. So now our formula looks like this:

```
IF( ISNULL( Close_Date__c ) ,
          value_if_true,
          value_if_false)
```

By replacing *`value_if_true`* and *`value_if_false`* with the other formulas we talked about, we've now figured out our whole formula:

```
IF( ISNULL( Close_Date__c ) ,
             TODAY()  -  Open_Date__c ,
             Close_Date__c  -  Open_Date__c )
```

Great! Our formula calculates the number of days a position has been open, regardless of whether it's currently open or closed. Now, let's go define a field for it on our Position object.

Try It Out: Defining a "Days Open" Custom Formula Field

We'll begin building the formula field the same way we created our other custom fields.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Formula` data type, and click **Next**.

Step 2 of the New Custom Field wizard appears.

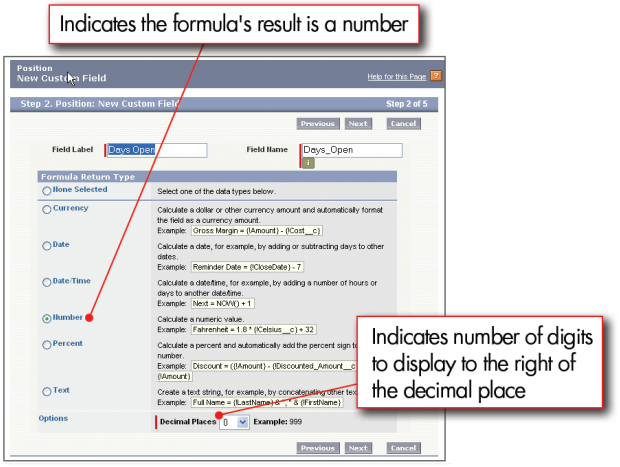


Figure 25: Custom Formula Field Wizard Step 2

5. In the `Field Label` field, enter Days Open.
6. Select the `Number` formula return type.

In this case, even though we're subtracting Date fields, we want to end up with just a regular numeric value.

7. Change the `Decimal Places` value to 0, and click **Next**.

Now it's time to enter the details of our formula.

8. Click the Advanced Formula tab, as shown in the following screenshot.

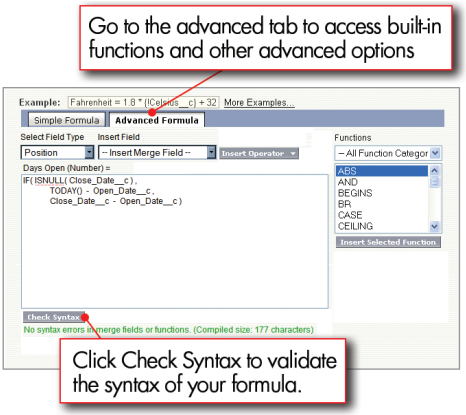


Figure 26: Custom Formula Field Editor

We want to use the Advanced Formula tab so we can access the platform's built-in functions through the Functions list on the right side.

9. From the `Functions` list, double-click `IF`.

Our formula now looks like this:

```
IF(logical_test, value_if_true, value_if_false)
```

Let's go ahead and define the logical test:

10. Delete *`logical_test`* from the formula, but leave your cursor there.
11. From the `Functions` list, double-click `ISNULL`.
12. Delete *`expression`* from the `ISNULL` function you just inserted, but leave your cursor there.
13. From the `Select Field Type` drop-down list, choose Position.
14. From the `Insert Field` drop-down list, choose `Close Date`.

Did you notice that you didn't have to remember to use the internal name of the `Close Date` field? The platform remembered for you when it inserted the value. Our formula now looks like this:

```
IF( ISNULL( Close_Date__c ) , value_if_true, value_if_false)
```

Now, let's specify the value if our logical test evaluates to true:

15. Delete *`value_if_true`* from the formula, but leave your cursor there.
16. Press Enter on your keyboard, and space over 10 spaces.

Adding the carriage return and spaces makes our formula more legible for others.

17. From the `Functions` list, double-click `TODAY`.
18. From the `Insert Operator` drop-down list, choose Subtract.
19. From the `Insert Field` list, choose `Open Date`.

We're getting closer—our formula now looks like this:

```
IF( ISNULL( Close_Date__c ) ,
          TODAY()  -  Open_Date__c , value_if_false)
```

Finally, let's specify the value if our logical test evaluates to false:

20. Delete `value_if_false` from the formula, but leave your cursor there.
21. Press Enter on your keyboard, and space over 10 spaces.
22. From the `Insert Field` list, choose `Close Date`.
23. From the `Insert Operator` drop-down list, choose Subtract.
24. From the `Insert Field` list, choose `Open Date`.

Our formula now matches our original:

```
IF( ISNULL( Close_Date__c ) ,
          TODAY()  -  Open_Date__c ,
          Close_Date__c  -  Open_Date__c )
```

Now that we've gone through those steps of the procedure, note that we could have just typed in the formula that we figured out in the last section. However, using the formula editor is a lot easier because you don't have to remember function syntax or internal names of fields and objects. Let's keep going and finish up this field:

25. Click **Check Syntax** to check your formula for errors.
26. In the `Description` text box, enter "The number of days a position has been (or was) open."
27. Add an optional `Help Text` description if you wish.
28. Select `Treat blank fields as blanks`, and click **Next**.
29. Accept all remaining field-level security and page layout defaults.
30. Click **Save**.

Try It Out: Giving Fields Dynamic Default Values

We can also use custom formulas to give our fields dynamic default values. While some fields like the `Travel Required` checkbox or the `Job Location` picklist have default values that

apply in every situation, there are other fields with defaults that can't be so easily defined. For example, at Universal Containers, recruiters are generally expected to fill a position within 90 days of it being opened. While we can't choose a single date that will always be 90 days after a position is opened, we *can* define a custom formula that takes the date the position is created and adds 90 days. The platform allows us to specify this formula as the `Hire By` field's default value:

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **Edit** next to the `Hire By` field.
4. Next to the `Default Value` text box, click **Show Formula Editor**.

Look familiar? This is the same editor that we used to define our `Days Open` custom formula field.

5. From the `Functions` list, double-click `TODAY`.
6. From the `Insert Operator` drop-down list, choose Add.
7. Type 90.

Your default value formula should be:

```
TODAY() + 90
```

8. Click **Save**.

It's that easy! Now to wrap up the fields on our Positions tab, let's set the default value of the `Open Date` field to the day that the record is created. To do this, follow these steps again, but use `TODAY()` as the `Default Value`.

Look at What We've Done

Let's revisit the Positions tab to take a look at what we've just done.

1. Click the Positions tab.
2. Click **New**.

Our `Days Open` formula field doesn't show up on our Position Edit page—that's because it's a formula field and doesn't require any user input to display. However, we can see that our `Open Date` and `Hire By` fields already have default values: `Open Date` should be today's date, and `Hire By` is 90 days later. We can change these values if we want, or we can just leave them as they are.

In order to see our `Days Open` field, we'll have to define our first position record. Let's do that now.

3. Enter any values you want to define a new position. At the very least, you must enter a value for the required `Position Title` field.
4. Click **Save**.

The new position is now displayed in its own record detail page. At the bottom of the page, notice our `Days Open` formula field, just above the `Created By` field. It should show 0, since we just created the position. If you want to see the value change, edit the record and set the `Open Date` to a week earlier. Isn't that neat?

Introducing Validation Rules

Now that we've defined all the fields that we want on our Position object, let's see if we can articulate a couple of rules about the data that should be entered into those fields. Even though the recruiters and hiring managers at Universal Containers are bright people, everyone sometimes makes mistakes when filling out a form, and a good app should catch the obvious errors.

For example, does it ever make sense for the value of the `Min Pay` field to be more than the value of the `Max Pay` field? Or should `Close Date` ever be unspecified if the `Status` field is set to Closed - Filled or Closed - Not Approved? Clearly not. We can catch these sorts of errors in our app with yet another built-in feature of the platform: *validation rules*.

Validation rules verify that the data a user enters in your app meets the standards that you specify. If it doesn't, the validation rule prevents the record from being saved, and the user sees an error message that you define either next to the problematic field or at the top of the edit page. Let's build a couple of validation rules now for our Recruiting app.

Note: You can find dozens of sample validation rules on the Successforce website at http://blogs.salesforce.com/features/2006/12/data_validation.html.

Try It Out: Defining a Validation Rule for Min and Max Pay

For our first validation rule, let's start off simple: `Min Pay` should never be greater than `Max Pay`:

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.

3. In the Validation Rules related list, click **New**.

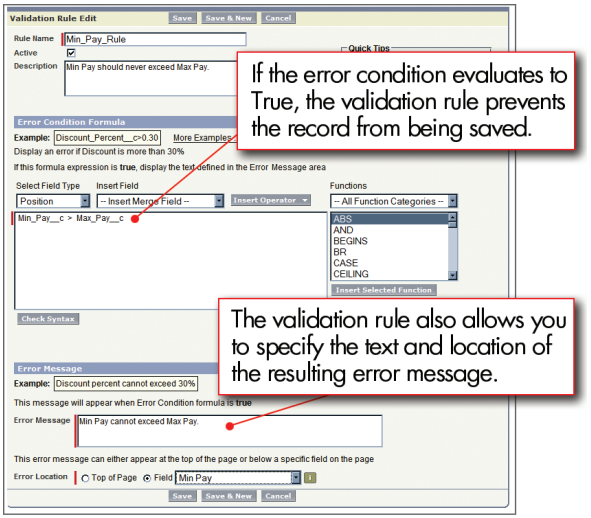


Figure 27: Validation Rule Edit Page

4. In the `Rule Name` text box, enter Min_Pay_Rule.

The name of a validation rule can't include any spaces, but if you forget, the platform helpfully changes them to underscores (_) for you.

5. Select the `Active` checkbox.

This checkbox specifies whether the validation rule should start working as soon as it's saved. Because this rule is pretty straightforward (and because we want to test it later!), it makes sense to turn it on right away.

6. In the `Description` text box, enter "Min Pay should never exceed Max Pay."

Now it's time to define the meat of our validation rule: the *error condition*. If you have a sense of déjà vu when looking at the Error Condition Formula area of the page, don't be alarmed! Just like formula fields and default field values, a validation rule can leverage a number of built-in operators and functions to define a true-or-false error condition that determines whether data is valid. When this condition evaluates to true, an error message displays and the record can't be saved.

We want our error condition to be true whenever `Min Pay` is greater than `Max Pay`, so let's use our formula editor to specify that now:

7. From the `Select Field Type` drop-down list, choose the Position object.

8. From the `Insert Field` drop-down list, choose the `Min Pay` field.
9. From the `Insert Operator` drop-down list, choose Greater Than.
10. From the `Insert Field` drop-down list, choose the `Max Pay` field.

You should now have an error condition formula that looks like this:

```
Min_Pay__c  >  Max_Pay__c
```

Now the only thing that remains is to specify the error message when our error condition evaluates to true.

11. In the `Error Message` text box, enter "Min Pay cannot exceed Max Pay."
12. Next to the `Error Location` field, select the `Field` radio button, and then choose `Min Pay` from the drop-down list.

Tip: Because our rule only requires a user to correct one field, our error message is displayed next to that field. If a rule requires a user to update multiple fields, it's more appropriate to place the error message at the top of the page.

13. Click **Save**.

Easy! Now that we've familiarized ourselves with a simple validation rule, let's define one that's a little trickier.

Try It Out: Defining a Validation Rule for Close Date

For our next validation rule, let's ensure that `Close Date` has a value whenever the `Status` field is set to Closed - Filled or Closed - Not Approved.

The hardest part of this validation rule is defining the error condition formula. When defining a condition like this, it's sometimes easiest to think about it in logical terms first, and then translate that logic to the functions and operators that are provided in the formula editor. In this case, our error condition is true whenever:

```
Close Date is Not Specified

AND

(Status is "Closed - Filled" OR
           "Closed - Not Approved")
```

Let's start with the first piece: "Close Date is Not Specified." To translate this into terms the formula editor understands, we'll need to use the `ISNULL()` function again. As you might

remember from defining the `Days Open` custom formula field, `ISNULL()` takes a single field or expression and returns true if it doesn't contain a value. So, remembering that we have to use the internal field name of the `Close Date` field in our formula, `Close Date is Not Specified` translates to:

```
ISNULL( CloseDate__c )
```

Next, let's figure out how to translate "Status is 'Closed - Filled'." To test for picklist values, we'll need to use another function: `ISPICKVAL()`. `ISPICKVAL()` takes a picklist field name and value, and returns true whenever that value is selected. So "Status is 'Closed - Filled'" translates to:

```
ISPICKVAL( Status__c , "Closed - Filled")
```

Now we just have to combine these translations with the functions for `AND()` and `OR()`. Both of them take an unlimited number of expressions, and 'AND' or 'OR' them all, respectively. For example:

```
AND ( exp1, exp2, exp3)
```

returns true when *`exp1`*, *`exp2`*, and *`exp3`* are all true. Likewise,

```
OR ( exp1, exp2, exp3)
```

returns true when any one of *`exp1`*, *`exp2`*, or *`exp3`* are true.

Put these functions all together with our other expression translations, and we get our completed error condition formula:

```
AND (
    ISNULL(  CloseDate__c ),
    OR (
         ISPICKVAL(  Status__c , "Closed - Filled"),
         ISPICKVAL(  Status__c , "Closed - Not Approved"))
)
```

Phew! Now we can quickly define our second validation rule using this formula:

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Validation Rules related list, click **New**.
4. In the `Rule Name` text box, enter Close_Date_Rule.
5. Select the `Active` checkbox.

6. In the `Description` text box, enter "Close Date must be specified when Status is set to 'Closed - Filled' or 'Closed - Not Approved.'"
7. In the Error Condition Formula area, enter the following formula:

```
AND (
    ISNULL(  Close_Date__c ),
    OR (
         ISPICKVAL(  Status__c , "Closed - Filled"),
         ISPICKVAL(  Status__c , "Closed - Not Approved"))
)
```

8. Click **Check Syntax** to make sure you didn't make a mistake.
9. In the `Error Message` text box, enter "Close Date must be specified when Status is set to 'Closed.'"
10. Next to the `Error Location` field, select the `Field` radio button, and then choose `Close Date` from the drop-down list.
11. Click **Save**.

Look at What We've Done

Let's revisit the Positions tab to test the validation rules that we've just made.

1. Click the Positions tab.
2. Click **New**.

First let's try defining a new position with a value for `Min Pay` that's larger than `Max Pay`.

3. Specify any value for the required `Position Title` field.
4. In the `Min Pay` field, enter 80,000.
5. In the `May Pay` field, enter 40,000.
6. Click **Save**.

Did you see what happened? An error message popped up that looked exactly like any other error message in the app!

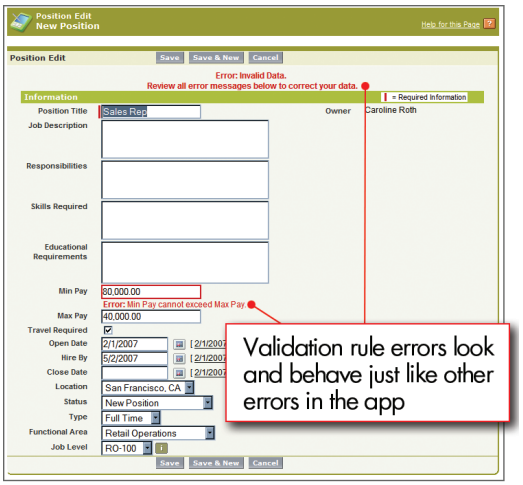


Figure 28: Error Message from a Validation Rule

Now let's test our other validation rule:

7. In the `Min Pay` field, enter 40,000.
8. In the `May Pay` field, enter 80,000.
9. From the `Status` drop-down list, choose Closed - Not Approved.
10. Click **Save**.

Our second validation rule is triggered, this time because we didn't specify a value for `Close Date`. Once we do, the record saves normally.

The Positions tab is now fully functional, with a couple of validation rules to ensure that users don't make certain mistakes. But are the fields where we want them? Are the fields that must have values marked as required? In the next section, we'll fine-tune our Position custom object by modifying its page layout.

Introducing Page Layouts

After defining all those fields and validation rules, we now have a fully functional Position custom object. However, it doesn't look all that nice—all of the long text areas appear at the top, and it's hard to scan. Let's move some things around to make this page easier for our users. We can do that by customizing the Position object's page layout.

A *page layout* controls the position and organization of the fields and related lists that are visible to users when viewing a record. Page layouts also help us control the visibility and editability

of the fields on a record. We can set fields as read-only or hidden, and we can also control which fields require users to enter a value and which don't.

Page layouts are powerful tools for creating a good experience for our users, but it's crucial that we remember one important rule: *page layouts should never be used to restrict access to sensitive data that a user shouldn't view or edit*. Although we can hide a field from a page layout, users can still access that field through other parts of the app, such as in reports or via the API. (We'll learn more about security that covers all parts of the app in *Chapter 7: Securing and Sharing Data* on page 109.)

Now let's see if we can organize the fields on our Position object in a way that's more user friendly.

Becoming Familiar with the Page Layout Edit Page

First let's take a look at the Page Layout Edit page:

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Page Layouts related list, click **Edit** next to the Position Layout.

Welcome to the Page Layout Edit page! As you can see, this editor is different from the ones that we've already used in other areas of the platform. That's because we're designing a user interface and need to see a representation of how our page will look as we're working. Before going any further, let's give ourselves a quick orientation to how this page is set up.

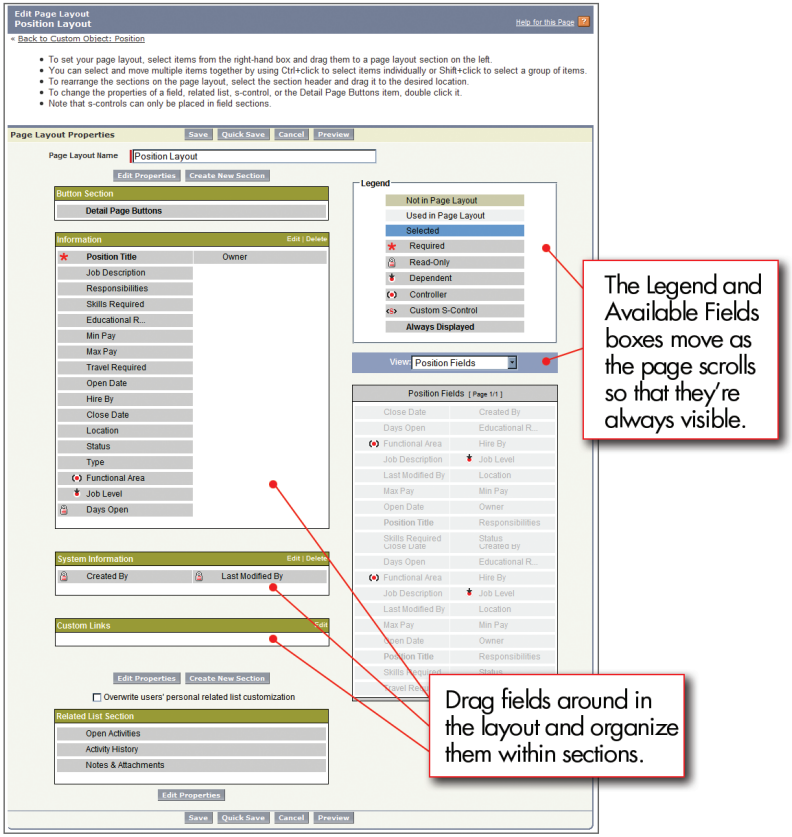


Figure 29: Page Layout Edit Page

On the left side of this page we can see a representation of what our page currently looks like. After an initial set of buttons, the Information section shows all of the fields we've created, plus an additional field for `Owner` that's in its own column. Below it is another section for System Information, then Custom Links, and finally, Related Lists.

As we scroll down to view the entire page, the boxes on the right side move with us so that they're always visible. These boxes include a legend for all of the icons we see on the left, plus a list of all of the Position fields, related lists, and other components that are available to put on our layout. Currently all of the fields in the box on the right are grayed out—that's because they've all been placed on our page layout by default. If we were to move one of the fields that's in the layout to the left back to the Position fields box on the right, that field would effectively be removed from the layout, and its name would no longer be grayed out.

Now that we know what we're looking at, let's rearrange the fields in the way a user might want to see them.

Try It Out: Grouping Fields into a New Section

Let's start modifying our page layout by first defining a new section for salary information. On a page layout, a section is simply an area where we can group similar fields under an appropriate heading. This makes it easy for our users to quickly identify and enter the information for a record, especially if our object has a large number of fields:

1. Click **Create New Section**.
2. In the `Name` text box, enter Compensation.

The `Name` field controls the text that's displayed as the heading for the section.

3. In the `Columns` drop-down list, choose 2 (Double).

This option allows us to choose whether we want the fields in our section to be arranged in two columns or one. The default is two columns and is the most commonly-used choice. However, if our section is going to contain text area fields, the one-column layout gives them more space on the page for display.

4. In the `Tab Order` drop-down list, choose Left-Right.

This setting controls the direction that a user's cursor will move when using the Tab key to navigate from field to field.

5. Select the options for `Show Section Heading on Detail Page` and `Show Section Heading on Edit Page`.
6. Click **OK**.

Voilà! We have a new section for Compensation just under the Custom Links related list. Let's move it above the System Information section and add the `Min Pay` and `Max Pay` fields:

7. Click the heading of the Compensation section and drag it above the System Information section.
8. Now use drag-and-drop to move the `Min Pay` and `Max Pay` fields from the Information section to the new Compensation section, as shown in the following screenshot.

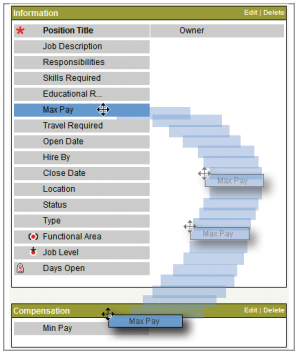


Figure 30: Dragging and Dropping Fields in a Page Layout

Now that we've gone through the process for building one section, let's build another:

9. Create a new one-column Description section below the Compensation section.
10. Drag `Job Description`, `Responsibilities`, `Skills Required`, and `Educational Requirements` into it.

Tip: You can use Shift+click or Ctrl+click to select all of these fields together and then drag them as one unit into the Description section.

Finally, to finish up our layout, let's reorganize the fields in the Information section so they're more readable.

11. In the first column, arrange these fields as follows:
    - Position Title
    - Status
    - Type
    - Functional Area
    - Job Level
    - Travel Required
12. In the second column, arrange these fields as follows:
    - Owner
    - Location
    - Open Date
    - Hire By
    - Close Date
    - Days Open

That's much better—our fields are organized, and it's easy to locate all of the information we need. Now all we have left to do is make the `Min Pay` and `Max Pay` fields required when a user defines a new position. Once we do that, we'll be all done with our Position object!

Try It Out: Editing Field Properties

Let's make the `Min Pay` and `Max Pay` fields required:

1. On the Page Layouts Edit page, double-click the `Min Pay` field.

This popup window allows us to edit the `Min Pay` field's properties. We can set the field to read-only and/or required:

- If it's read-only, a user who views a Position record edit page won't be able to change its value.
- If it's required, a user won't be able to create a Position record without specifying a value.

If we didn't want a user to see the `Min Pay` field at all, we could simply drag it off the layout and onto the Position Fields box on the right side.

Caution: Don't forget our earlier warning! Page layouts should never be used to restrict access to sensitive data that a user shouldn't view or edit. That's because page layouts control only a record's edit and detail pages; they don't control access to fields in any other part of the platform.

2. Select the `Required` checkbox, and click **OK**.
3. Repeat these steps for the `Max Pay` field.
4. Click **Save** to finish customizing the page layout.

Hooray! We're all done with our Position object's page layout.

Look at What We've Done

Congratulations. We've just built ourselves a simple Recruiting app that tracks details about an organization's open job postings. Let's check out what we've done by revisiting the Positions tab and clicking **New**. Because of the changes that we've made in our page layout, our Position edit page should now look like this:

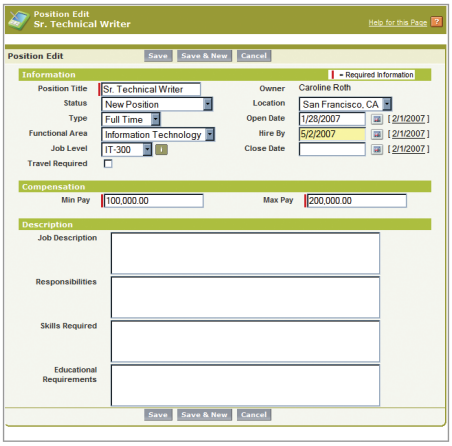


Figure 31: Final Version of the Position Edit Page

We have an object with a tab, we've added custom fields, and we've arranged them in a page layout. We've finished our simple app, and now we're well on our way to creating the more complex Recruiting app that we described earlier.

Now things are going to get even more interesting. In the next chapter, we'll add a few more custom objects to track things like candidates, job applications, and reviews, and then we'll enhance our Recruiting app even further by defining how our objects relate to one another. Before you know it, we're going to have an incredibly powerful tool, all implemented with a few clicks in the platform.

Chapter 6

Expanding the Simple App Using Relationships

In this chapter ...

So far we've accomplished a fair amount—we've created the Recruiting app and built out a fully functional Position custom object with a tab and several types of fields. It's a good start, but there's more to do.

Having just one object in our Recruiting app is like having a party with just one guest—not all that interesting! We need to invite more "people" to the party by building custom objects to represent candidates, job applications, and reviews, and, even more importantly, we need to create *relationships* between them. Just like a party isn't all that fun if you don't know any of the other guests, an app isn't all that powerful unless its objects have links to other objects in the app. That's going to be the focus of this chapter, so let's get started!

Introducing Relationships

So what is a relationship, and why are they important for our app? Just as a personal relationship is a two-way association between two people, in terms of relational data, a relationship is a two-way association between two objects. Without relationships, we could build out as many custom objects as we could think of, but they'd have no way of linking to one another.

For example, after building a Position object and a Job Application object, we could have lots of information about a particular position and lots of information about a particular candidate who's submitted an application for it, but there would be no way of seeing information about the job application when looking at the Position record, and no way of seeing information about the position when looking at the Job Application record. That's just not right!

With relationships, we can make that connection and display data about other related object records on a particular record's detail page. For example, once we define a relationship between the Position and Job Application objects we just talked about, our Position record can have a related list of all the job applications for candidates who have applied for the position, while a Job Application record can have a link to the positions for which that candidate is applying. Suddenly the "people" at our Recruiting app "party" know some of the other guests, and the app just got a lot more interesting.

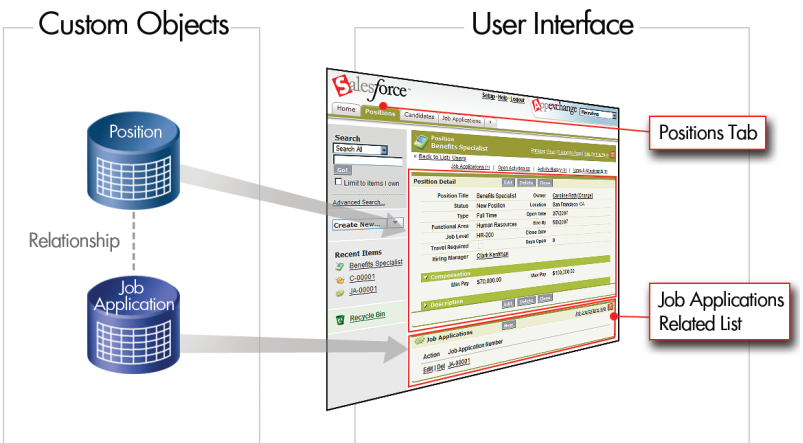


Figure 32: Relationships Allow Information about Other Object Records to be Displayed on a Record Detail Page

Introducing Lookup Relationship Custom Fields

As we learned in *Chapter 3: Reviewing Database Concepts* on page 23, we can define a relationship between two objects through the use of common fields. On the platform, we can define relationships between objects by creating a *lookup relationship* custom field from one object to another. A lookup relationship field is a custom field on an object record that contains a link to another record. When we place a lookup relationship custom field on an object, we're effectively creating a many-to-one relationship between the object on which the lookup relationship field is placed and the other object.

For example, if we placed a lookup relationship field on a Job Application object that referenced Position object records, many Job Application records could be related to a single Position record. This would be reflected both with a new Position field on the Job Application record page and with a new Job Applications related list on the Position record detail page.

That's the sort of thing that we're going to do in this chapter. First, let's start with the really quick and easy example of putting a `Hiring Manager` field on our Position object—we'll create a many-to-one relationship between the Position object and the standard User object that comes with every organization, reflecting the fact that a hiring manager can be responsible for several positions at a time. Then we'll build out a few more objects and implement a more complex relationship involving Positions, Job Applications, and Candidates.

Try It Out: Relating Hiring Managers to Positions

For our first relationship, let's associate a hiring manager with a position by putting a lookup relationship field on the Position object. The lookup field will allow users to select the hiring manager for the position by selecting from all the users of the Recruiting app.

For example, if Ben Stuart, our recruiter, wants to assign Anastasia O'Toole as the hiring manager for the Benefits Specialist position, he'll be able to do so by clicking the lookup icon () next to the lookup relationship field that we create. Her name will then appear on the Position detail page.

To create the lookup relationship field that accomplishes this, we'll need to go back to the now familiar Position object detail page.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select `Lookup Relationship`, and click **Next**.
5. In the `Related To` drop-down list, choose User, and click **Next**.

As we've mentioned, User is a standard object that comes with all organizations on the platform. It contains information about everyone who uses the app in your organization.

6. In the `Field Label` text box, enter Hiring Manager. Once you move your cursor, the `Field Name` text box should be automatically populated with Hiring_Manager.
7. Click **Next**.
8. Accept the defaults in the remaining two steps of the wizard.
9. Click **Save**.

Look at What We've Done

Now return to the Positions tab, and click **New**. The Position Edit page includes a new `Hiring Manager` lookup field! If you click the lookup icon next to this field (), you can search through all of the users of the Recruiting app and select one as the hiring manager. That user's name now appears on the Position record:

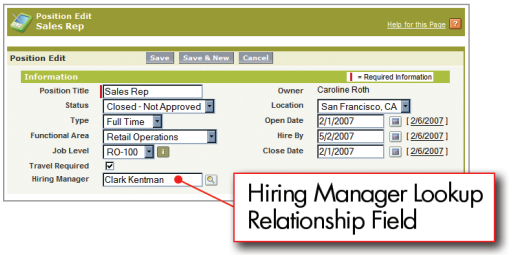


Figure 33: Hiring Manager Lookup Relationship

As you can see, it was easy to set up this simple relationship between Positions and Users. And as a general rule, you'll find that relationships are pretty easy to set up.

What gets a little tricky is when we start wanting to create relationships that don't represent a simple one-to-one or many-to-one relationship. We'll see an example of one of those in a little bit. Right now, let's build a custom object for candidates so we'll be able to create some more relationships in our Recruiting app.

Adding Candidates to the Mix

Let's add a Candidate custom object to our app so we can manage the information about our candidates. We'll also add fields to the object, modify the page layout properties, and create a

Candidate record. The process for creating the Candidate custom object is almost identical to the one we followed to create the Position custom object, so we'll zip through this quickly.

Try It Out: Creating the Candidate Object

To create our Candidate custom object, navigate back to **Setup ➤ Build ➤ Custom Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 8: Values for Defining the Candidate Object

Field	Value
Label	Candidate
Plural Label	Candidates
Object Name	Candidate
Description	Represents an applicant who might apply for one or more positions
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Candidate Number
Data Type	Auto Number
Display Format	C-{00000}
Starting Number	00001
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	Yes

To create the Candidates tab, select a `Tab Style` in the first step of the wizard, and then accept all the defaults until you get to the Add to Custom Apps page. On this page, select only the Recruiting App, and then click **Save**.

The Recruiting app now has three tabs: Home, Positions, and Candidates. Now let's add some custom fields to the Candidate object.

Try It Out: Adding Fields to the Candidate Object

To create custom fields on the Candidate object, click **Setup ➤ Build ➤ Custom Objects**, and click **Candidate** to view its detail page. In the Custom Fields & Relationships related list, use the **New** button to create custom fields according to the following table. Where necessary, we've indicated some additional values you'll need to fill in. Otherwise, you can simply accept all defaults.

One difference you'll see in the Candidate object fields is that three of them—First Name, Last Name, and Email—have the `External ID` option selected. This option allows the values in these fields to be indexed for search from the sidebar of the application. If we didn't select these values as external IDs, we'd only be able to search for records based on the `Candidate Number` field. Setting the `Email` field as an external ID is also going to help us with importing data a little later in this chapter.

Note: There are a lot of custom fields on the Candidate object. If you don't care about having your app match the screenshots in this book, you only need to bother with the first four fields in the following table.

Table 9: Candidate Object Custom Fields

Data Type	Field Label	Other Values
Text	First Name	Length: 50 External ID: Selected
Text	Last Name	Length: 50 External ID: Selected
Phone	Phone	
Email	Email	External ID: Selected
Text	Street	Length: 50

Data Type	Field Label	Other Values
Text	City	Length: 50
Text	State/Province	Length: 50
Text	Zip/Postal Code	Length: 15
Text	Country	Length: 50
Text	Current Employer	Length: 50
Number	Years of Experience	Length: 2 Decimal Places: 0
Text	SSN	Length: 9
Picklist	Education	Picklist values: • HS Diploma • BA/BS • MA/MS/MBA • Ph.D. • Post Doc
Checkbox	Currently Employed	Default: Checked
Checkbox	US Citizen	Default: Checked
Checkbox	Visa Required	Default: Unchecked
Phone	Mobile	
Phone	Fax	

Try It Out: Modifying the Candidate Page Layout Properties

To finish up with this object, let's organize all of our fields on the page layout and mark some fields as required. To do so, let's go to the Page Layout Properties page.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Candidate**.
3. In the Page Layouts related list, click **Edit** next to the Candidate Layout.

4. Create three new double-column sections below the Information section: Address, Employment, and Additional Details. Drag the appropriate fields into them, as shown in *Figure 34: Candidate Object Page Layout* on page 88, and don't forget to click **Quick Save** so you can save your work as you go.
5. Set the `First Name`, `Last Name`, and `Email` fields to required:
    a. Use Ctrl+click to select all three required fields.
    b. Click the **Edit Properties** button.
    c. Select the `Required` checkbox in the Select All row, and click **OK**.
6. Click **Save**.

Your Page Layout Properties page should now look similar to the following screenshot.

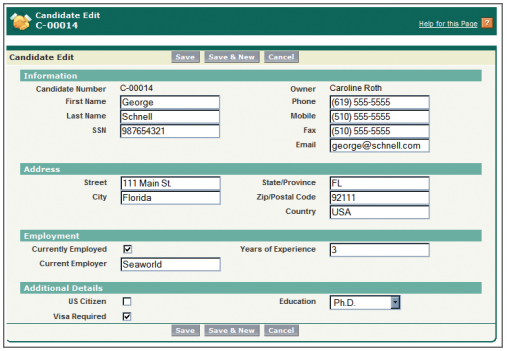


Figure 34: Candidate Object Page Layout

Look at What We've Done

Here's a quick way to verify that you did everything correctly.

1. Click the Candidates tab.
2. Click **New**.
3. Create a new record for a candidate named Ethan Tran.
4. Enter a value for each of the required fields.
5. Click **Save**.

How does the page layout look? Are the fields where you want them? If you were able to successfully create a new Candidate record, and everything looks okay, let's move on—we have more relationships to create!

Creating a Many-to-Many Relationship

Now that we've got two custom objects in our Recruiting app, the only logical thing that remains for us to do is relate them, right?

Of course! However, unlike the simple relationship that we created between hiring managers and positions, the relationship between positions and candidates is going to be a little more complex.

Why? Let's think about how these two objects relate to one another:

- A candidate can apply to many positions.
- A position can have many candidates apply for it.

Instead of a many-to-one relationship, such as the one that exists between positions and hiring managers, we've got a many-to-many relationship. We want a candidate to be able to apply for multiple positions and a position to have multiple candidates, but our lookup relationship field only allows us to create a one-to-many relationship. What would the "one" side be?

Here's where we get a little creative. Earlier in the chapter you might recall us talking about a Job Application object. A job application fits into the space between candidates and positions—one candidate can submit many job applications, and one position can receive many job applications, but a job application always represents a request from a single candidate for a single position. In essence, the Job Application object has a many-to-one relationship with both the Candidate object and the Position object.

The Job Application object is the key to making a many-to-many relationship. Much like flour, yeast, salt, and water can be combined to form a loaf of bread, we can combine two lookup relationships and a *junction object* like Job Application to make a many-to-many relationship. Even though we don't have a way of directly relating candidates and positions, the Job Application object brings them together.

For example, let's look at a typical scenario at Universal Containers. Ethan is a candidate who applies for the Project Manager position and the Sr. Developer position. Bonnie is a candidate who applies for the Sr. Developer position. Every time Ethan or Bonnie apply for a position, a Job Application record tracks the exact position for which they're applying. As you can see in the following diagram, Ethan can apply to many positions, and Bonnie and Ethan (or many candidates) can apply to the same position.

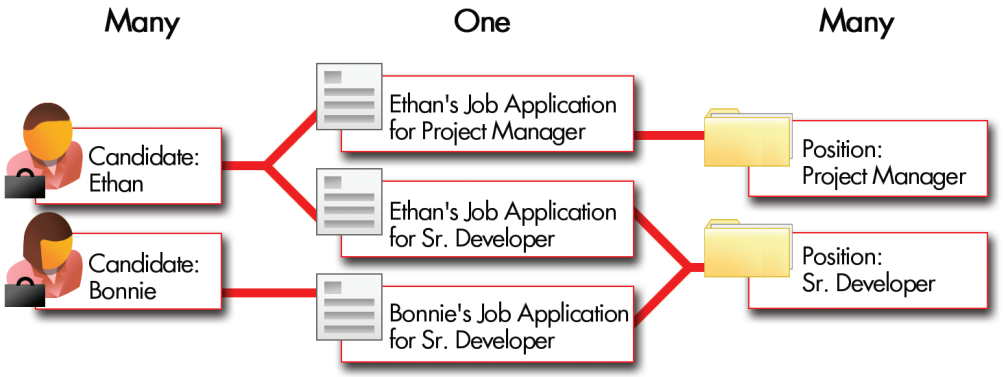


Figure 35: Using a Job Application Object to Create a Many-to-Many Relationship Between Candidates and Positions

In relational database terms, each Job Application record is a row in the Job Application table, consisting of a foreign key to a Candidate record and a foreign key to a Position record. The following entity relationship diagram shows this relationship.

Figure 36: Entity Relationship Diagram for the Candidate, Job Application, and Position Objects

Consequently, in order to define a many-to-many relationship between the Candidate and Position objects, we'll need to create a Job Application object with the following additional fields:

- A Candidate lookup relationship
- A Position lookup relationship

Let's go do that now.

Try It Out: Creating the Job Application Object

You should be a pro at this by now! To create our Job Application custom object, navigate back to **Setup ➤ Build ➤ Custom Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 10: Values for Defining the Job Application Object

Field	Value
Label	Job Application
Plural Label	Job Applications
Object Name	Job_Application
Description	Represents the junction object between a candidate and a position
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Job Application Number
Data Type	Auto Number
Display Format	JA-{00000}
Starting Number	00001
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	Yes

To create the Job Applications tab, select a `Tab Style` in the first step of the wizard, and then accept all the defaults until you get to the Add to Custom Apps page. On this page, select only the Recruiting App, and then click **Save**.

That was simple enough, but we're not quite done. We need to create the lookup relationship fields between the Job Application object and the Position and Candidate objects.

Try It Out: Adding Fields to the Job Application Object

Here's another procedure that we've done several times before, but this time we only need to define three custom fields instead of the nearly twenty that we built for the Candidate object. We'll need to add the two lookup relationship fields that will create relationships between the Job Application object and the Position and Candidate objects, and we'll also need to add a picklist field so that we can track the application's status.

Although these fields are almost identical to the ones we created earlier, you'll notice when you're defining the lookup relationship fields that there's a new step in the custom field wizard: Step 6: Add Custom Related Lists. This step of the wizard is where we can specify a heading for the Job Applications related list that will show up on both the Candidate and Position detail pages when we add it to their page layouts.

Why didn't we see this step earlier when we created our `Hiring Manager` lookup field? It turns out that because User is a standard object without a tab or a detail page of its own, we'll never have a need for a Positions related list. The platform knows this, so it leaves out the related list step whenever someone adds a lookup relationship field that references the User object.

Now that we're all squared away with that small difference, let's finish up these Job Application fields. Click **Setup ➤ Build ➤ Custom Objects**, and then click **Job Application** to view its detail page. In the Custom Fields & Relationships related list, use the **New** button to create custom fields according to the following table. Where necessary, we've indicated some additional values you'll need to fill in. Otherwise you can simply accept all defaults.

Table 11: Add Custom Fields to the Job Application Object

Data Type	Field Label	Other Values
Lookup Relationship	Candidate	Related To: Candidate Related List Label: Job Applications
Lookup Relationship	Position	Related To: Position Related List Label: Job Applications
Picklist	Status	Picklist values: • New

Data Type	Field Label	Other Values
		• Phone Screen • Schedule Interviews • Extend an Offer • Hired • Rejected Use first value as default value: Selected

Look at What We've Done

Voilà! If you click on the new Job Applications tab and click **New**, you'll see the `Candidate` lookup field, a `Position` lookup field, and a `Status` picklist field.

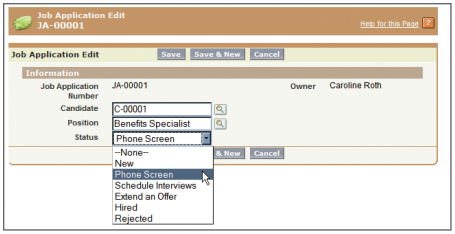


Figure 37: Custom Fields on the Job Application Edit Page

But there's more! Because we've built a couple of lookup relationships, our Candidate and Position record detail pages now each have a new Job Applications related list. And the Job Application detail page includes links to the Candidate and Position records that it references. All of our objects are now related and linked to one another!

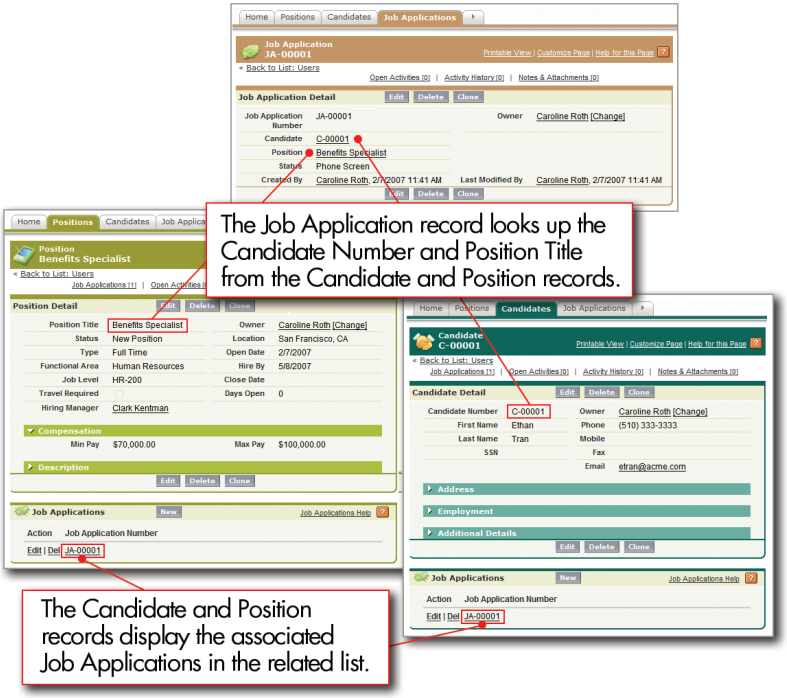


Figure 38: Job Application Links to Position and Candidate Data

Before we move on to our last custom object, let's see if we can clean up the usability of our app a bit so our users don't have to identify candidates and job applications by number when they click the lookup button in the Job Application edit page, or when they look at the Job Applications related list on the Candidate or Position detail pages.

Introducing Search Layouts

By default, all lookup dialogs and related lists that result from new relationships such as the ones we've defined in this chapter only display the record name or number. For example, if you went ahead and created a job application, you might have found the Candidate lookup dialog a little cryptic because the only listed field is `Candidate Number`, as shown in the following diagram.

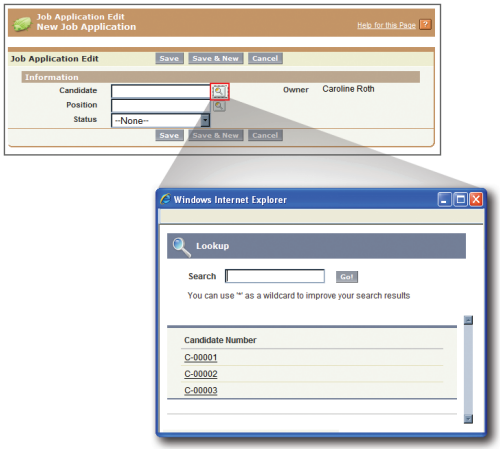


Figure 39: Default Candidate Lookup on the Job Application Object

Likewise, the Job Applications related lists on the Position and Candidate detail pages only display a job application number. It would be much more useful if these related lists also included the associated candidate's name or position.

To fix these issues, we can add fields to the *search layouts* for the objects that we've defined. Search layouts are ordered groups of fields that are displayed when a record is presented in a particular context, such as in search results or a related list. By adding fields, we can give users more information and help them locate records more quickly.

The Search Layouts related list on the custom object detail page is the place to modify these sets of fields. Go to **Setup ➤ Build ➤ Custom Objects** and select the Candidates object. You'll see that the available search layouts include:

Table 12: Available Search Layouts

Layout Name	Description
Search Results	Search results that originate from searching in the left sidebar of the application or in advanced search
Lookup Dialogs	Lookup dialog results that originate from clicking the button ( ) next to a lookup field on an edit page
Candidates Tab	The list of recent records that appears on the home page of a tab, and in related lists on other object detail pages
Search Filter Fields	The filters that can be applied to search results

Note: The List View layout also appears in the Search Layouts related list, but it's not for specifying fields. Instead, it allows you to specify the buttons that appear on the list view page for an object.

Try It Out: Adding Fields to the Candidate Lookup Dialog

First let's add fields to our Candidate lookup dialog:

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Candidate**.
3. In the Search Layouts related list, click **Edit** next to the Lookup Dialogs layout.

The Edit Search Layout page includes a list of available fields from the Candidate object. You can choose any number of fields to include in the lookup dialog, and order them in any way you choose.

4. Move the following fields into the Selected Fields box under Candidate Number:
   - First Name
   - Last Name
   - City
   - State/Province
   - Phone
5. Click **Save**.

That's it! To try it out, return to the Job Applications tab, and click **New**. When you click the `Candidate` lookup button, the dialog is now much more useful.

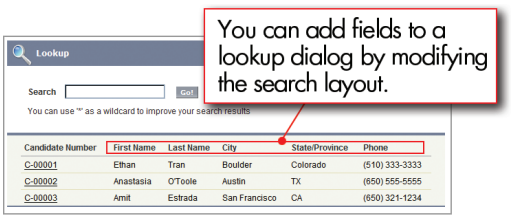


Figure 40: Modified Candidate Lookup on the Job Application Object

Try It Out: Updating Additional Search Layouts

Now that we've updated one search layout for lookups, the rest should be easy. Use the Search Layouts related list on the custom object detail page to modify the other search layouts as described in the following table.

Table 13: Additional Search Layouts

Object	Search Layout	Add These Fields
Candidate	• Search Results • Candidates Tab	• Candidate Number • First Name • Last Name • City • State/Province • Phone
Candidate	• Search Filter Fields	• Candidate Number • First Name • Last Name • Education • Years of Experience • City • State/Province • Country • Currently Employed
Position	• Search Results • Lookup Dialogs • Positions Tab • Search Filter Fields	• Position Title • Location • Functional Area • Job Level • Type • Hiring Manager • Status • Open Date • Close Date

Object	Search Layout	Add These Fields
Job Application	• Search Results • Lookup Dialogs • Job Applications Tab • Search Filter Fields	• Job Application Number • Candidate • Position • Status • Created Date • Owner First Name • Owner Last Name

Now we're nearly finished building all the objects and relationships that we need for our Recruiting app. We simply need to create one more custom object to provide our hiring managers and interviewers with a place to enter their comments about candidates.

Managing Review Assessments

Interviewers, recruiters, and hiring managers need to be able to create reviews so that they can record their comments about each candidate. They also need to see the reviews posted by other people, as well as make comments on them if they want to add details. To allow our users to perform these tasks, we'll need to create a custom Review object and relate it to the Job Application object.

The Review object has a many-to-one relationship with the Job Application object because one job application can have one or more reviews associated with it. Once again, we'll use a lookup relationship field on the object to create the relationship. A related list on the Job Application will show the associated reviews, representing the "many" side of the relationship.

Figure 41: Review Has a Many-to-One Relationship with Job Application

Try It Out: Creating the Review Object

For the last time, let's create a new custom object. Navigate back to **Setup ➤ Build ➤ Custom Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 14: Values for Defining the Review Object

Field	Value
Label	Review
Plural Label	Reviews
Object Name	Review
Description	Represents an interviewer's assessment of a particular candidate
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Review Number
Data Type	Auto Number
Display Format	R-{000000}
Starting Number	000001
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	No

Notice that we didn't launch the tab wizard this time. Reviews don't need a tab of their own, because they can be accessed via a related list on the Job Application detail page. When you create an object with a tab, the platform provides access to that object's records in various places other than just the tab, such as in search results and the Recent Items list in the sidebar area

of every page. Because most Recruiting app users won't need to see reviews unless it's in the context of a job application, we don't need to create a separate tab for them.

Now let's finish up the custom fields on the Review object.

Try It Out: Adding Fields to the Review Object

For our Review object we only need to build two fields: a lookup relationship field to the Job Application object and a text area field for entering the reviewer's assessment.

Click **Setup ➤ Build ➤ Custom Objects**, and then click **Review** to view its detail page. In the Custom Fields & Relationships related list, use the **New** button to create custom fields according to the following table. Where necessary, we've indicated some additional values you'll need to fill in. Otherwise, you can simply accept all defaults.

Table 15: Add Custom Fields to the Review Object

Data Type	Field Label	Other Values
Lookup Relationship	Job Application	Related To: Job Application Related List Label: Reviews
Text Area (Long)	Assessment	Length: 32,000 # of Visible Lines: 6

Try It Out: Customizing the Review Object's Page and Search Layouts

First let's update the page layout of the Review object so that the `Assessment` text field is in a single column section of the same name.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Review**.
3. In the Page Layouts related list, click **Edit** next to Review Layout.
4. Click **Create New Section** and define a new one-column section named Assessment.
5. Move the Assessment section just below the Information section, and put the `Assessment` field in it.
6. Click **Save**.

Now let's configure our review search layouts so that reviews are always displayed with the associated job application and author.

7. In the Search Layouts related list on the Review object detail page, click **Edit** next to Lookup Dialogs and add the following fields:
   - Review Number
   - Job Application
   - Owner First Name
   - Owner Last Name
   - Created Date
8. Repeat for the Search Filter Fields layout.

To update the Reviews related list that appears on the Job Application detail page, we'll have to edit the related list directly on the Job Application page layout. This is different from how we added fields to the Job Application related list on the Position and Candidate detail pages because the Review object doesn't have an associated tab, and therefore doesn't have a tab search layout. Remember—the tab search layout is responsible for both the fields that appear in the list on the tab home page and the default fields that appear in related lists on other object detail pages.

Note: The tab search layout is responsible for the fields in the related list layout only if the related list properties have not been modified on other objects' page layouts. For example, if you modify the properties of the Job Application related list on the Position page layout, those changes will always override the field specifications of the Job Application tab search layout.

Because the Review object doesn't have a tab search layout, we have to set those fields another way.

9. Click **Setup ➤ Build ➤ Custom Objects**.
10. Click **Job Application**.
11. In the Page Layouts related list, click **Edit** next to Job Application Layout.
12. Scroll down to the Related List Section, select `Reviews` and click **Edit Properties**.
13. Add the following fields to the Selected Fields box:
    - Review Number
    - Owner First Name
    - Owner Last Name
    - Created Date

14. From the `Sort By` drop-down list, choose Review Number.
15. Click **OK**.
16. Click **Save** on the page layout edit page.

Look at What We've Done

Terrific! Let's go see what we've made:

1. Click the Job Applications tab and select a record, or create one if you haven't already.

Tip: When you use the Candidate and Position lookup dialogs as you're creating a Job Application record, note that, by default, they only display the most recently viewed records. You can locate additional records by using the search box, which returns records based on the `Candidate Number` or `Position Title` fields, respectively.

Use the * wildcard with other characters to improve your search results. For example, searching on C* returns every Candidate record. Likewise, searching on *e returns all position records that include the letter 'e' in the title.

After the job application is created, notice that the Reviews related list now appears on the Job Application detail page. That's because we related the Review object to the Job Application object with a lookup relationship.

2. In the Reviews related list, click **New** to create a review.

Do you see how the platform automatically filled in the job application number in the review's edit page? That's one of the small, but important benefits of using the platform to build an application like this—not only is it easy to create links and relationships between objects, but the platform anticipates what we're doing and helps us accomplish our task with as few clicks as possible.

Now go ahead and click around the rest of the app, creating a few more positions, job applications, candidates, and reviews. Pretty neat, huh? Our data is all interconnected, and our edits to the search layouts allow us to view details of several related objects all at once.

Putting it All Together

We just created several objects and a lot of relationships. The following simple diagram shows us what we've accomplished so far.

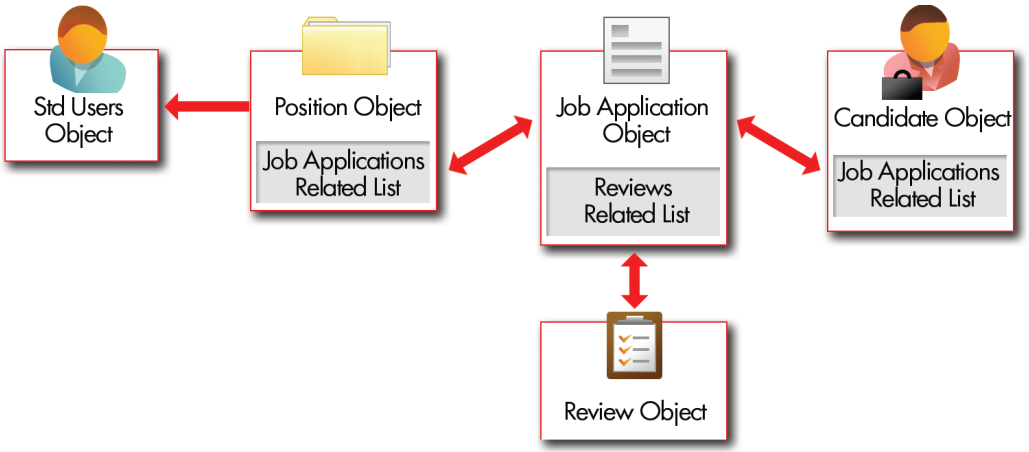


Figure 42: Recruiting App Relationships

If you're interested, you can take a look at the following entity relationship diagram.

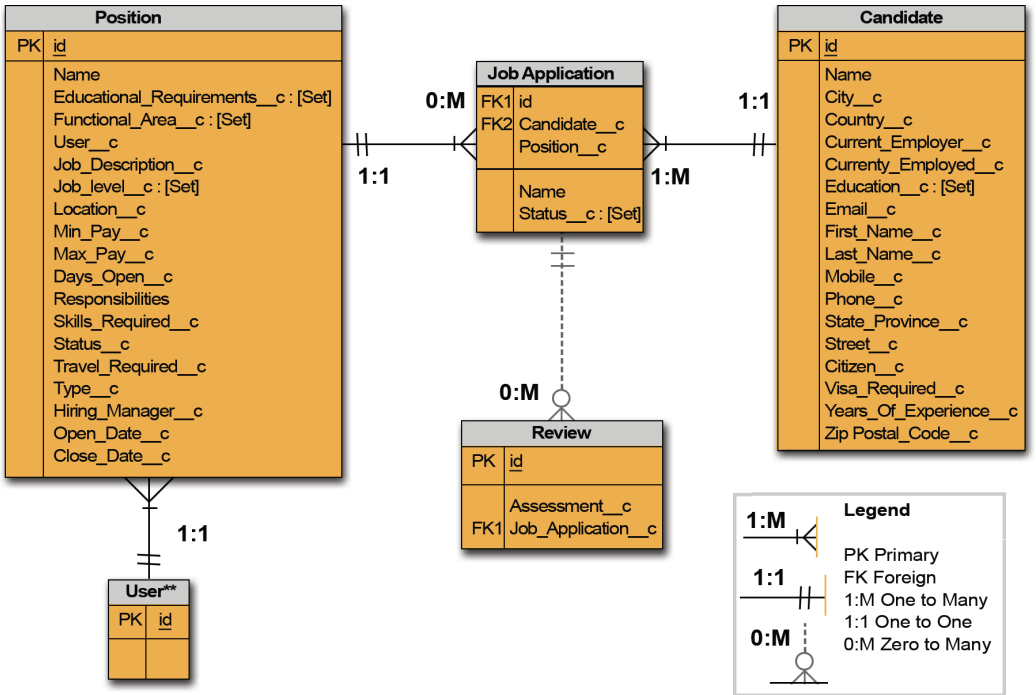


Figure 43: Recruiting App Entity Relationship Diagram

We've now built all of our Recruiting app objects and tabs, and we've defined lots of custom fields—everything from text fields and picklists to more complex formula fields and lookup

relationship fields. We've created a robust user interface so that our recruiters and hiring managers can enter and retrieve data about positions and related candidates, job applications, and reviews, and we did all of this without writing a single line of code!

Remember when we assigned Clark Kentman as the hiring manager for the Benefits Specialist position? Let's look at what Clark can do now: He can create and update his positions, and he can view other positions. He can look at details about any candidates who have applied for the Benefits Specialist job, and he can review their related job applications. He can also check the status of the job applications. He no longer has to go to Human Resources to search through Microsoft Word documents and spreadsheets to manage his tasks in the hiring process. The Recruiting app is well on its way to becoming a fully-functional and useful application!

However, before we leave this chapter behind, let's get ourselves prepared for the rest of this book by creating and importing some real data. It'll help us when we get to our next chapter on security and sharing if we have some records that we can work with.

Try It Out: Downloading Sample Data

In addition to entering data via our tabbed pages, we can also use the handy Import Wizard to import multiple records at a time. The ability to easily import data into your custom objects is one of the Force.com platform's key benefits. Let's download some sample data so we can add more records to our custom objects without tons of typing.

1. Download the `RecruitingApp-2_0.zip` file containing the sample CSV (comma-separated values) import files from http://wiki.apexdevnet.com/index.php/Creating_On-Demand_Apps.
2. Extract the zip file to `C:\dev\recruiting` (or any directory on your computer).
3. Go to `C:\dev\recruiting`. This directory contains three CSV files: `Positions.csv`, `Candidates.csv`, and `JobApplications.csv`. (The directory also contains one other file that you'll use later when we build composite components for our app in *Chapter 10: Moving Beyond Native Apps* on page 225.)

Before we import anything, we need to make a modification to the import file for Positions. The sample `Positions.csv` you downloaded contains fictional users in the Hiring Manager column. The names of these users most likely won't match any user in your organization, and if you import the file "as is," the Import Wizard won't be able to find any matching users, and the `Hiring Manager` field on each Position record will be left blank. So let's go ahead and make that change.

4. Go to `C:\dev\recruiting`, and open `Positions.csv` in Excel, a text editor, or any other program that can read CSV files.

5. In the Hiring Manager column, replace the fictional users with the first and last name of a user in your organization.
6. Save the file, making sure to maintain the CSV format.

Note: If your locale isn't English (United States), the date and field values in `Positions.csv` are also invalid. You'll need to change them before you import.

Try It Out: Using the Import Wizard

Now let's walk through the process of importing Position records using the Import Wizard and the `Positions.csv` file you downloaded.

1. Click **Setup ➤ Data Management ➤ Import Custom Objects**.
2. Click **Start the Import Wizard!**. The Import Wizard appears.
3. Select `Position` for the type of record you're importing, and click **Next**.
4. Choose `Yes` to prevent duplicate Position records from being created as a result of this import. Accept the other defaults for matching, and click **Next**.
5. Select `None` for the record owner field. We didn't include a User field in the CSV file to designate record owners. The Import Wizard will assign you as the owner of all new records.
6. Choose the `Hiring Manager` lookup relationship field so you can link Position records with existing User records in the Recruiting app, and click **Next**.
7. Select `Name` as the field you want to match against as the Import Wizard compares `Hiring Manager` names in your import file with User names in the system, and click **Next**.
8. Click **Browse**, and find `C:\dev\recruiting\Positions.csv`. Click **Next**.
9. Use the drop-down lists to specify the Salesforce fields that correspond to the columns in your import file. For your convenience, identically matching labels are automatically selected. Click **Next**.
10. Click **Import Now!**.

Use the following table to repeat the import process for Candidate records. You'll notice the wizard skips the two steps about lookup relationship field matching—because the Candidate object doesn't have any lookup relationship fields, the Import Wizard automatically leaves those steps out.

Table 16: Importing the `Candidates.csv` File

For this wizard step...	Select these options...
1. Choose Record	Candidate
2. Prevent Duplicates	No—insert all records in my import file
3. Specify Relationships	None
4. File Upload	Browse to `C:\dev\recruiting\Candidates.csv`
5. Field Mapping	Accept all defaults
6. Verify Import Settings	Click **Import Now!**

Finally let's do it one more time for Job Application records. In this iteration, we're going to make use of the `Email` field, an external ID on the Candidate object, to match up job applications with the correct candidate records.

Table 17: Importing the `Job_Applications.csv` File

For this wizard step...	Select these options...
1. Choose Record	Job Application
2. Prevent Duplicates	No—insert all records in my import file
3. Specify Relationships	`Which user field...?` None `Which lookup fields...?` Candidate, Position
4. Define Lookup Matching	`Which field on Candidate...?` Email (External ID) `Which field on Position...?` Position Title
5. File Upload	Browse to `C:\dev\recruiting\Job_Applications.csv`
6. Field Mapping	`Email (col 0)`: Candidate `Position Title (col 1)`: Position
7. Verify Import Settings	Click **Import Now!**

Great! While the files are importing, you can go to **Setup ➤ Monitoring ➤ View the Import Queue** to check on their status.

Once the import operations have completed, return to the Positions, Candidates, or Job Applications tab and click **Go!** next to the `View` drop-down list. You'll see a list of all the new records you just imported.

We've just added a bunch of data to our app without a lot of work. In the next chapter, we'll take a look at all the ways we can control access to this data using the built-in tools of the platform. We'll get into the nitty-gritty about security, sharing rules, permissions, roles, and profiles.

Chapter 7

Securing and Sharing Data

In this chapter ...

Now that we've got all of our object relationships in place, it's time to start thinking about who's actually going to be using the app and how much access they should have to all of the data that it's going to contain.

As with many apps, our Recruiting app exposes sensitive pieces of data such as social security numbers, salary amounts, and applicant reviews that could really come back to haunt us if they fell into the hands of the wrong people. We need to provide security without making it harder for our recruiters, hiring managers, and interviewers to do their jobs.

Here we're going to see another one of the huge benefits that Force.com has to offer—simple-to-configure security controls that easily allow us to restrict access to data that users shouldn't see, without a lot of headaches. Similar to Access Control Lists or Windows folder permissions, Force.com allows us to specify who can view, create, edit, or delete any record or field in the app. In this chapter, we'll see how we can use Force.com to implement those rules.

Controlling Access to Data in Our App

As we've already seen, there are three types of users who will need to access the data in our Recruiting app: recruiters, hiring managers, and interviewers. To these three, let's add a fourth type of user—a standard employee who doesn't perform any interviews and who never needs to hire anyone. (This employee will help us determine the default permissions that should apply to all of the new recruiting objects in our app.)

One by one, let's take a look at the kinds of access that each one of these users needs and, more importantly, the kinds of access they *don't* need to do their jobs. Once we've compiled a set of required permissions, we'll figure out how to implement them in the rest of the chapter.

Required Permissions for the Recruiter

For our first set of required permissions, let's take a look at Mario Ruiz, a recruiter at Universal Containers. To do his job, Mario needs to be able to create, view, and modify any position, candidate, job application, or review that's in the system. Likewise, Mario needs to view and modify the recruiting records that all of the other recruiters own, since all of the recruiters at Universal Containers work together to fill every position, regardless of who created it.

Although Mario has the most powerful role in our Recruiting app, we still can't give him complete free reign. State and federal public records laws require that all recruitment-related records must be saved for a number of years so that if a hiring decision is questioned, it can be defended in court. Consequently, we need to make sure that Mario will never accidentally delete a record that needs to be saved to fulfill the law.

But how will he keep the number of positions, candidates, job applications, and reviews in check if he can't delete? Won't the app become swamped with old data? Not if we're smart about it—instead of having Mario delete old records, we can use the `Status` field on a record as an indication of whether it's current. We'll filter out all of the old records by using a simple list view.

Here's a summary of the required permissions that we need to implement for a recruiter:

Table 18: Summary of Required Permissions: Recruiter

	Position	Candidate	Job Application	Review
Recruiter	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit

Required Permissions for the Hiring Manager

Our next set of required permissions is more challenging. Ben Stuart, our hiring manager, needs to be able to access the recruiting records related to his open positions, but he shouldn't be mucking around with other recruiting records (unless they're owned by other hiring managers who report to him). Also, there are certain sensitive fields that he has no need to see, like the social security number field. Let's go object by object to really drill down on what Ben does and doesn't need to access in order to perform his job:

Position

First of all, Ben likes to post his own positions so that he can publicize them as fast as possible, but in our app, Mario the recruiter ultimately needs to take ownership of the record to make sure the position gets filled. As a result, Ben needs the ability to create positions, but then we'll need to find a mechanism to make sure that they ultimately get transferred to Mario for ownership. (Hint: as you'll see in *Chapter 8: Using Custom Workflow and Approval Processes* on page 153, we'll tackle that problem with a workflow rule that transfers position ownership to a recruiter when a new position is created by a hiring manager. For now just assume that this already works.)

Ben should also be able to update and view all fields for positions for which he's the hiring manager, but he should only be able to view other managers' positions.

Candidate

Ben sometimes wants to poach a prime candidate who's applying for a position under another manager, but this is a practice that Universal Containers frowns upon. As a result, Ben should only be able to view those candidates who have applied for a position on which he's the hiring manager. Also, since Ben has no reason to see a candidate's social security number, this field should be restricted from his view.

Job Application

Because Ben is restricted to viewing only those candidates who have applied for a position on which he's the hiring manager, he should likewise view only the job applications that relate those candidates to his open positions. As the hiring manager, he needs to be able to update the status of those job applications to specify which candidates should be selected or rejected.

Review

To make a decision about the candidates who are applying, Ben needs to see the reviews posted by the interviewers, as well as make comments on them if he thinks the interviewer was being too biased in his or her review.

Likewise, Ben needs to be able to create reviews so that he can remember his own impressions of the candidates he interviews.

Here's a summary of the required permissions we need to implement for a hiring manager:

Table 19: Summary of Required Permissions: Hiring Manager

	Position	Candidate	Job Application	Review
Hiring Manager	• Read • Create • Edit*	• Read* (No SSN)	• Read* • Edit*	• Read • Create • Edit

* Only for those records that are associated with a position to which the hiring manager has been assigned

Required Permissions for the Interviewer

For our third set of required permissions, let's take a look at Melissa Lee's role as an interviewer. Ben, her manager, likes Melissa to interview candidates for highly technical positions, but doesn't want her speaking with folks who are applying for roles on the user interface team. As a result, Melissa should be able to view only the candidates and job applications to which she's assigned as an interviewer. No such restriction needs to exist on the positions that are out there, but she shouldn't be able to view the minimum and maximum salary values for any of them. Likewise, she shouldn't see the social security number of any candidate, since it's sensitive information that has nothing to do with her job.

Melissa must be able to create and edit her reviews so that she can record her comments about each candidate, but she shouldn't be able to see the reviews of other interviewers—reading them might sway her opinion one way or the other. As with hiring managers and recruiters, Melissa also shouldn't be allowed to delete any records to ensure that public records laws are fulfilled.

Here's a summary of the required permissions we need to implement for an interviewer:

Table 20: Summary of Required Permissions: Interviewer

	Position	Candidate	Job Application	Review
Interviewer	• Read (No min/max pay)	• Read* (No SSN)	• Read*	• Read** • Create

	Position	Candidate	Job Application	Review
				• Edit**

* Only for those records that are associated with a position to which the interviewer has been assigned

** Only for those records that the interviewer owns

Required Permissions for the Standard Employee

Employees, such as Manny Damon on the Western Sales Team, are often the best resources for recruiting new hires, even if they are not active hiring managers or interviewers. For this reason, we need to make sure that employees like Manny can view open positions, but that they can't see the values for the positions' minimum and maximum salary fields—otherwise they might tip off friends to negotiate for a position's maximum salary value! Manny also shouldn't be able to view any other records in our Recruiting app.

Here's a summary of the required permissions we need to implement for a standard employee:

Table 21: Summary of Required Permissions: Standard Employee

	Position	Candidate	Job Application	Review
Standard Employee	• Read (No min/max pay)	-	-	-

So Where Are We Now?

Now that we've gone through the required permissions for each of our four users, let's organize our thoughts by summarizing them in the following table. In the rest of this chapter, we'll figure out how we can use the platform to implement these rules in our Recruiting app.

Table 22: Summary of Required Permissions

	Position	Candidate	Job Application	Review
Recruiter	• Read • Create	• Read • Create	• Read • Create	• Read • Create

	Position	Candidate	Job Application	Review
	• Edit	• Edit	• Edit	• Edit
Hiring Manager	• Read • Create • Edit*	• Read* (No SSN)	• Read* • Edit*	• Read • Create • Edit
Interviewer	• Read (No min/max pay)	• Read* (No SSN)	• Read*	• Read** • Create • Edit**
Standard Employee	• Read (No min/max pay)	-	-	-

* Only for those records that are associated with a position to which the hiring manager/interviewer has been assigned

** Only for those records that the interviewer owns

Tip: When implementing the security and sharing rules for your own organization, it's often useful to create a required permissions table like this to organize your thoughts and make sure you don't forget to restrict or grant access to a particular user. You'll see that we're going to refer back to this table again and again as we go through this chapter.

Data Access Concepts

Before we get started implementing our security and sharing rules, let's quickly take a look at all the ways that we can control data on the platform:

Object-Level Security

The bluntest way that we can control data is by preventing a user from seeing, creating, editing, and/or deleting any instance of a particular type of object, like a Position or Review. Object-level access allows us to hide whole tabs and objects from particular users, so that they don't even know that type of data exists.

On the platform, we set object-level access rules with object permissions on user profiles. We'll learn more about profiles in a little bit.

Field-Level Security

A variation on object-level access is field-level access, in which a user can be prevented from seeing, editing, and/or deleting the value for a particular field on an object. Field-level access allows us to hide sensitive information like the maximum salary for a position or a candidate's social security number without having to hide the whole object.
On the platform, we set field-level access rules with the field-level security. We'll also learn more about that shortly.

Record-Level Security

To control data with a little more finesse, we can allow particular users to view an object, but then restrict the individual object records that they're allowed to see. For example, record-level access allows an interviewer like Melissa Lee to see and edit her own reviews, without exposing the reviews of everyone else on her team.
On the platform, we actually have four ways of setting record-level access rules:

- *Organization-wide defaults* allow us to specify the baseline level of access that a user has in your organization. For example, we can make it so that any user can see any record of a particular object to which their user profile gives them access, but so that they'll need extra permissions to actually edit one.
- *Role hierarchies* allow us to make sure that a manager will always have access to the same records as his or her subordinates.
- *Sharing rules* allow us to make automatic exceptions to organization-wide defaults for particular groups of users.
- *Manual sharing* allows record owners to give read and edit permissions to folks who might not have access to the record any other way.

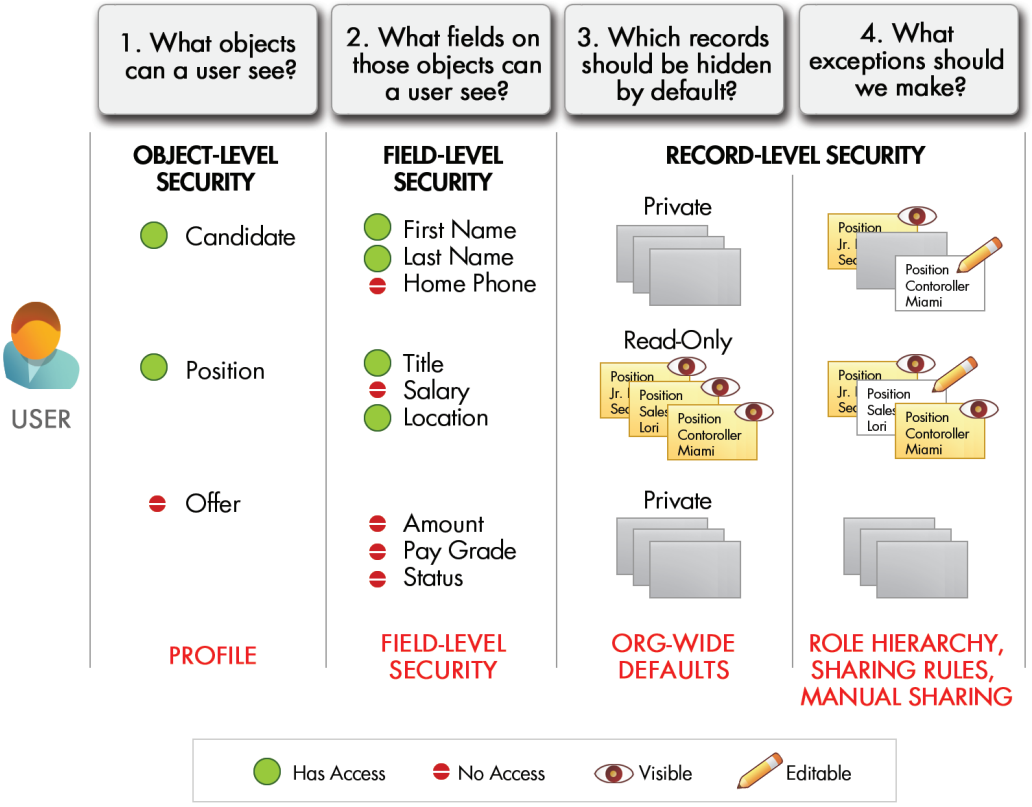


Figure 44: Controlling Data with Force.com

The combination of all of these sharing and security settings in platform means that we can easily specify user permissions for an organization of thousands of users without having to manually configure the permissions for each individual. Pretty neat! Now let's get started learning more about each of these methods for controlling data, and actually implementing the security and sharing rules for our app.

Controlling Access to Objects

First let's configure access to our Recruiting app custom objects. As we mentioned previously, we can control whether a user knows that a particular object exists in the app by modifying his or her profile. But exactly what is a profile, and what does it control?

Introducing Profiles

A *profile* is a collection of settings and permissions that determine what a user can do in the platform, kind of like a group in a Windows network, where all of the members of the group have the same folder permissions and access to the same software. Profiles control:

- The objects the user can view, create, edit, and delete
- The object fields the user can view and edit (more on that later!)
- The tabs the user can view in the app
- The standard and custom apps the user can access
- The page layouts a user sees
- The record types available to the user
- The hours during which the user can log in to the app
- The IP addresses from which the user can log in to the app

Profiles are typically defined by a user's job function (for example, system administrator or sales representative), but you can have profiles for anything that makes sense for your organization. A profile can be assigned to many users, but a user can be assigned to only one profile at a time.

Standard Profiles

The platform provides the following set of standard profiles in every organization:

- Read Only
- Standard User
- Marketing User
- Contract Manager
- Solution Manager
- System Administrator

Each of these standard profiles includes a default set of permissions for all of the standard objects available on the platform. For example, users assigned to the Standard User profile can never create, edit, or delete a campaign.

When a custom object is created, all standard profiles automatically get full access to the object by default (except, of course, for the Read Only profile, which gets only read access). The only way to restrict access to a custom object for users assigned to a standard profile is by defining field- or record-level access rules, which we'll learn more about later.

You can find more detailed descriptions of all the standard profiles in the online help, but the important thing to know is that you can never actually edit the permissions on a standard profile. Instead, if you have access to the Enterprise, Unlimited, or Developer Editions of the platform, you can make a copy of a standard profile and then customize that copy to better fit the needs of your organization. That's what we're going to end up doing for our Recruiting app (and as a result, Enterprise, Unlimited, and Developer Editions will be the only editions that the Recruiting app will support).

Custom Profiles in Our Recruiting App

For our app, we've talked about four types of users: recruiters, hiring managers, interviewers, and standard employees. We might just jump the gun and say that this equals four different user profiles, but let's take a closer look.

Recruiters are pretty straightforward—they definitely represent a particular job function, and they need access to different types of data than other users. They need their own profile.

A hiring manager, however, is not exactly a single type of position. For most organizations, a hiring manager in the Sales department will almost certainly need access to a different type of data than a hiring manager in Engineering. However, for the purposes of our app, sales managers and software managers still need the same types of access to recruiting data—reviews, candidates, positions, and job applications. Let's keep this as a single profile for now, but if incorporating our app into an organization with other CRM functionality, we'll need to suggest as a best practice that the hiring manager permissions for recruiting-related data need to be replicated for any profile to which hiring managers belong.

Finally, let's look at interviewers and standard employees. Neither one of these user types reflects a particular job function, and when you think about it, just about anyone in an organization might be called upon to perform an interview. Let's define a single profile for a standard employee and find a way to grant interviewers access to the records that they need through some other mechanism. (Hint: we can use a combination of organization-wide defaults and sharing rules to make this work.)

Try It Out: Creating the Recruiter Profile

All right—we're finally ready to dig into the app and create our first profile! Let's start with the Recruiter profile.

1. Click **Setup ➤ Manage Users ➤ Profiles**.

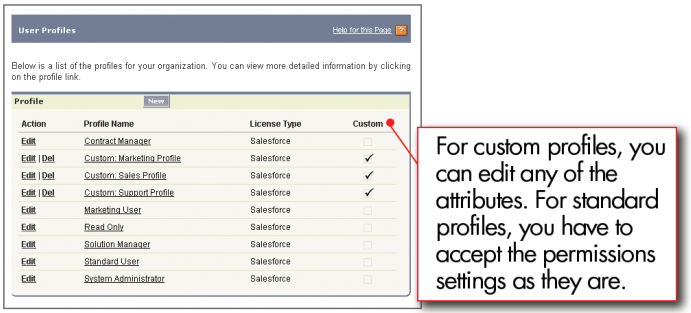


Figure 45: Standard Profiles

Here you should see the list of standard profiles that we talked about earlier. After we create our custom profiles, they'll also show up in this list.

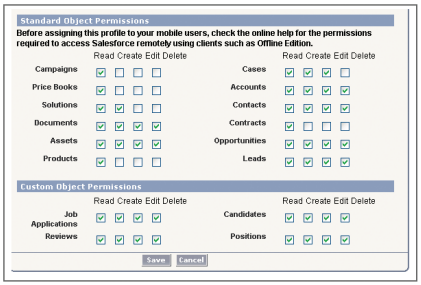
First, we can quickly tell which profiles we can play with by looking at the `Custom` column—if it's checked, that means it's a custom profile and we can edit anything about it. If that column is not checked, we can still click the **Edit** link; we just can't modify any of the permission settings. (What does that leave for us to edit on a standard profile? Well, we can choose which tabs should appear at the top of a user's page, and we can also select the apps that are available in the AppExchange app menu in the top-right corner of the page.)

2. Create a new profile named Recruiter based on the Standard User profile.

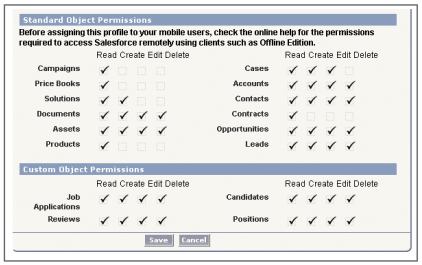
There are actually two ways of doing this—we can either click **New**, select an existing profile to clone, name it, and click **Save**, or we can simply click **Clone** in the detail page of the profile that we want to copy, name it, and click **Save**. Ultimately, it's the same number of clicks, so choose the method you like best. Standard User is the profile that most closely resembles what we want our new Recruiter profile to look like, so it's a good starting point.

3. In the new Recruiter profile's detail page, click **Edit**.

The Recruiter detail page should look and function exactly like the Standard User profile edit page except with one important difference: when you click **Edit**, you have the ability to modify any of the permission settings.



Permissions on Recruiter Profile - Editable



Permissions on Standard User Profile - Not Editable

Figure 46: Standard Versus Custom Profile Permissions Edit Page

4. In the Custom App Settings area, make the Recruiting app visible to users assigned to the Recruiter profile, as shown in the following screenshot.

Figure 47: Profile Custom App Settings Area

Tip: You can also give this profile access to any of the other available apps as well. Every profile needs to have at least one visible app.

When an app is visible, a user can select it from the AppExchange app menu at the top-right corner of the page. Be aware, however, that even if an app is visible, the app's tabs won't show up unless a profile has permissions to view the tabs and permission to view the associated object. (We'll set both of those permissions lower down in the Profile edit page.)

5. Select **Default** next to the Recruiting app.

Making this selection means that the Recruiting app will be displayed when a user logs in. You'll notice that when you select an app as the default, its `Visible` checkbox is automatically selected, because it doesn't make sense for an app to be the default if it's not visible to the user.

6. In the Tab Settings area, select Default On for the Positions, Job Applications, and Candidates tabs.

Tip: You can choose whether you want other tabs to be displayed based on the additional apps that you made visible in the last step.

For the purposes of our Recruiting app, all of our custom recruiting tabs are on by default. For any other tabs that you select, you can choose which should be displayed on top of the user's

page (Default On), hidden from the user's page but available when he or she clicks the All Tabs tab on the far right (Default Off), or completely hidden from the user (Tab Hidden).

Realize that even if you completely hide a tab, users can still see the records that would have appeared in that tab in search results and in related lists. (To prevent a user from accessing data, we have to set the proper restrictions in the Standard and Custom Object Permissions areas lower down in the Profile edit page—we'll get there shortly!)

The `Overwrite users' personal tab customizations` setting appears if you have an organization that's currently in use and you want to make sure your existing users are viewing the tabs that you've selected. You don't need to select this for our app, because we're defining a brand-new profile and no one has personalized his or her tab visibility settings yet. However, if you do want to select this option at some point in the future, just make sure you're not going to annoy your users by deleting all of their customizations!

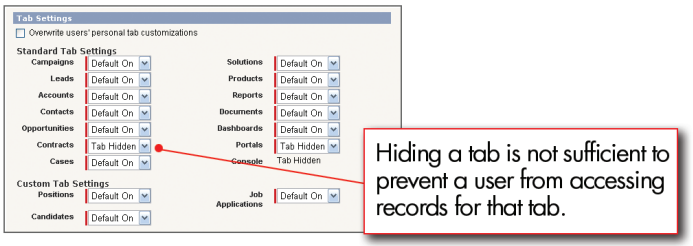


Figure 48: Profile Tab Settings Area

Just below the Tab Settings area, the Administrative and General User Permissions areas of the profile allow you to grant special access to features and functionality that don't map directly to particular objects. None of these permissions affects our Recruiting app, but you can learn more about them in the online help.

7. In the Custom Object Permissions area, specify the object-level permissions for our Recruiter profile according to the following table.

Table 23: Summary of Required Permissions: Recruiter

	Positions	Candidates	Job Applications	Reviews
Recruiter	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit

Tip: Depending on the apps that you made visible previously, you can also set additional object permissions on standard or other custom objects.

Since there are no instances when a recruiter should be allowed to delete a recruiting-related object, we should make sure that the object-level permissions for deletion are turned off on Positions, Candidates, Job Applications, and Reviews.

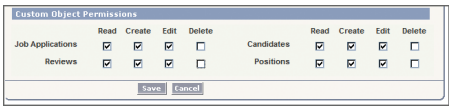


Figure 49: Recruiter Profile Permissions

By restricting the power to delete recruiting-related objects here, recruiters will *never* be able to delete these objects. However, the fact that we're granting recruiters permission to create, read, or edit our recruiting objects does not necessarily mean that recruiters will be allowed to read or edit *every* recruiting object record. Why?

Here we see the result of two really important concepts in the platform:

- The permissions on a record are always evaluated according to a combination of object-, field-, and record-level permissions.
- When object- versus record-level permissions conflict, the most restrictive settings win.

What this means is that even though we are granting this profile create, read, and edit permissions on the recruiting objects, if the record-level permissions for an individual recruiting record prove to be more restrictive, those will be the rules that will define what a recruiter can access.

For example, our new profile gives a recruiter permission to create, edit, and view reviews. However, if we set organization-wide defaults for reviews to Private (a record-level permission), our recruiter will be allowed to edit and view only his own reviews, and not the reviews owned by other users. We'll learn more about record-level permissions later and go through more examples of how they work with the object-level ones, but for now, just understand that object-level permissions are only one piece of the puzzle.

8. Click **Save** to create your profile and return to the profile detail page.

Congratulations! We're done with our first profile. As you can see, it really wasn't that hard, because we'd already analyzed our required permissions and knew what objects recruiters will need access to. In the next section, let's quickly finish up our other two profiles and then move on to field-level security.

Try It Out: Creating More Profiles

Now that we've created our Recruiter profile, let's finish up with profiles for hiring managers and standard employees. As we mentioned earlier, these profiles are probably too generic to be used in a company that's using the platform for other, non-recruiting functionality, but for our purposes, they'll work just fine for now.

To make each profile, go ahead and follow the steps that we outlined in the previous section. The important things to remember for these profiles are:

- The profiles should be named Hiring Manager and Standard Employee and should be based on the standard profile that best fits your needs. We used Standard User for our Recruiter profile, and that's probably a good one for Standard Employee as well. However, for your organization you might find that Hiring Manager more closely resembles the Contract or Solution Manager profile instead. It's up to you.
- The Recruiting app for both profiles must be set to `Visible`.
- The tabs for Positions, Candidates, and Job Applications should be set to Default On.
- Standard and custom object permissions should reflect the required permissions that we worked on before.

To refresh our memories, let's take a look at our required permissions summary table:

Table 24: Summary of Required Permissions: Hiring Manager, Interviewer, and Standard Employee

	Positions	Candidates	Job Applications	Reviews
Hiring Manager	• Read • Create • Edit*	• Read* (No SSN)	• Read* • Edit*	• Read • Create • Edit
Interviewer	• Read (No min/max pay)	• Read* (No SSN)	• Read*	• Read** • Create • Edit**
Standard Employee	• Read (No min/max pay)	-	-	-

* Only for those records that are associated with a position to which the hiring manager/interviewer has been assigned

** Only for those records that the interviewer owns

This table is a little confusing right now, because Interviewer and Standard Employee are still separated into two different users. Let's go ahead and combine them into a single Standard Employee user so that it's a little easier to see what object-level permissions we need to grant. It's easy to do, because Interviewers and Standard Employees have the same permissions on the Position object, and we already have asterisks on Candidates, Job Applications, and Reviews that ensures these users won't look at anything to which they're not assigned as an interviewer:

Table 25: Summary of Required Permissions: Combining Interviewers and Standard Employees

	Positions	Candidates	Job Applications	Reviews
Hiring Manager	• Read • Create • Edit*	• Read* (No SSN)	• Read* • Edit*	• Read • Create • Edit
Standard Employee (Interviewer)	• Read (No min/max pay)	• Read* (No SSN)	• Read*	• Read** • Create • Edit**

* Only for those records that are associated with a position to which the hiring manager/standard employee has been assigned

** Only for those records that the standard employee owns

Great, but what about all of those asterisks and restrictions on visible fields? Do we need to take those into account when we set our object-level permissions?

Not at all. Those asterisks and field restrictions represent record- and field-level security settings that we're going to have to specify elsewhere in our app. The only things we need to care about here are the permissions that these users will need to have access to at least *some* of the time—that's the whole point of object-level permissions:

- For hiring managers, that's create, read, and edit on Positions and Reviews, read on Candidates, and read and edit on Job Applications
- For standard employees, that's read on Positions, Candidates, and Job Applications, and create, read, and edit on Reviews

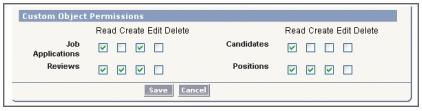


Figure 50: Hiring Manager Profile Permissions

Figure 51: Standard Employee Profile Permissions

Fantastic! We've just finished defining profiles and object-level permissions for all the users in our Recruiting app. However, that's still just one piece of the security and sharing puzzle—we still need to make sure that sensitive data on these objects is protected from users who don't need access, and then we need to drill down on the actual records that each user should be allowed to view and edit.

Controlling Access to Fields

Now that we've restricted access to objects as a whole, it's time to use a finer-toothed comb to manage the security of individual object fields. These are the settings that allow us to protect sensitive fields such as a candidate's social security number without having to hide the fact that the candidate object even exists.

Introducing Field-Level Security

In the platform, we control access to individual fields with *field-level security*. Field-level security controls whether a user can see, edit, and delete the value for a particular field on an object.

Unlike page layouts, which only control the visibility of fields on detail and edit pages, field-level security controls the visibility of fields in any part of the app, including related lists, list views, reports, and search results. Indeed, in order to be absolutely sure that a user can't access a particular field, it's important to use the field-level security page for a given object to restrict access to the field. There are simply no other shortcuts that will provide the same level of protection for a particular field.

Field-Level Security in Our Recruiting App

Just to refresh our memories about what field-level security settings we need for our Recruiting app, let's take another look at our required permissions in the following table. We'll keep them organized by recruiter, hiring manager, and standard employee, because it turns out that (surprise!) field-level security settings are closely related to profiles:

Table 26: Revised Summary of Required Permissions

	Position	Candidate	Job Application	Review
Recruiter	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit
Hiring Manager	• Read • Create • Edit*	• Read* (No SSN)	• Read* • Edit*	• Read • Create • Edit
Standard Employee	• Read (No min/max pay)	• Read* (No SSN)	• Read*	• Read** • Create • Edit**

* Only for those records that are associated with a position to which the hiring manager/standard employee has been assigned

** Only for those records that the standard employee owns

For field-level security settings, we need to zero in only on those rules that include field restrictions in parentheses, specifically:

- On Positions, hide minimum and maximum pay from standard employees
- On Candidates, hide social security numbers from hiring managers and standard employees

So let's get down to it!

Accessing Field-Level Security Settings

So where do you access field-level security settings? It turns out that there are actually two ways to get to those settings—either through a profile's detail page, or by choosing **Setup ➤ Security Controls ➤ Field Accessibility**.

The path that you choose depends on whether you're focused solely on field-level security, or whether you want to fiddle with page layout settings at the same time. The first way is the simplest and requires the least number of clicks, but the second is sometimes more convenient when you're building a new app, because you can perform two related tasks in the same page.

Because we have two different field rules that we need to implement for our Recruiting app, we'll define one rule with the first method, and the other with the second.

Try It Out: Restricting Access to a Position's Minimum and Maximum Salary Fields

Let's get started actually implementing the first of our field-level security rules: *On Positions, hide minimum and maximum pay from standard employees*. We'll define this rule by accessing field-level security settings through the Standard Employee profile's detail page.

1. Click **Setup ➤ Manage Users ➤ Profiles**, and then select the Standard Employee profile.

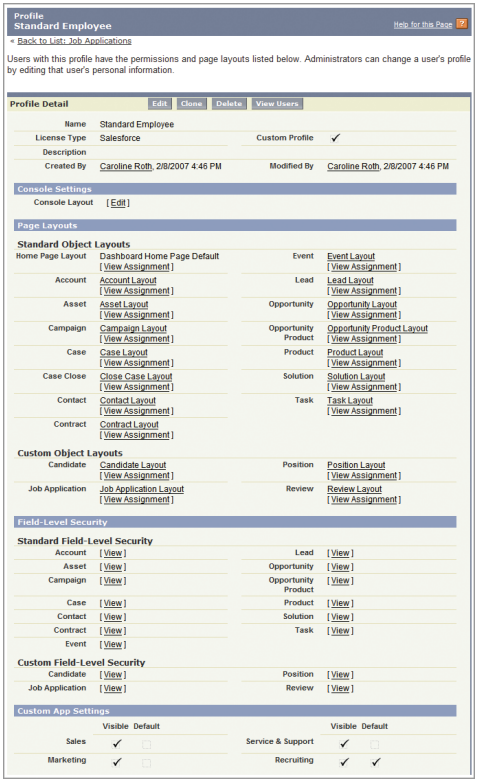


Figure 52: Standard Employee Profile Detail Page

The first thing you'll notice about the Standard Employee profile's detail page is that it includes several more areas than the edit page that we originally used to define the profile. These additional areas include Page Layouts (which we learned about in *Chapter 5: Enhancing the Simple App with Advanced Fields and Page Layouts* on page 55), Field-Level Security, Record Type Settings, Login Hours, and Login IP Ranges. Although we won't go into detail here about how to use any areas other than Field-Level Security, they're part of what makes a profile so powerful in our application. You can learn more about them in the online help.

2. In the Field-Level Security area, click **View** next to the Position object.
3. Click **Edit**.

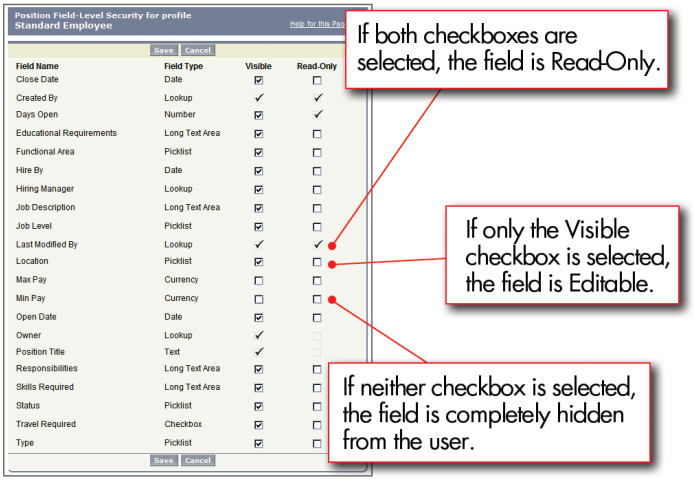


Figure 53: Field-Level Security Edit Page

Here we see security settings for all of the fields on Position, including `Min Pay` and `Max Pay`, the two fields that we want to restrict. You'll notice that some field-level security settings on some fields cannot be modified—this is because either they are system-generated fields or they act as lookup relationship fields (foreign keys) to other records.

Since the security settings checkboxes can be a little bit confusing, let's do a quick exercise to map their values (`Visible` and `Read-Only`) to the three logical permission settings for a field: "Hidden," "Read Only," and "Editable":

Table 27: Field-Level Permission Mappings

Permission	Visible	Read-Only
Hidden		
Read Only	X	X
Editable	X	

After doing this exercise, it's easy to see that most fields are editable, because their `Visible` checkbox is the only one selected. To restrict a field from ever being viewed by a user, all we have to do is deselect both checkboxes.

4. Next to the `Max Pay` field, deselect `Visible`.
5. Next to the `Min Pay` field, deselect `Visible`.
6. Click **Save**.

We're done!

Try It Out: Restricting Access to a Candidate's Social Security Number

For our second field-level security setting (*On Candidates, hide social security numbers from hiring managers and standard employees*), let's implement it using the Field Accessibility page. You'll see that this method is very similar to the last, but that the Field Accessibility page gives us more power and control over how we define our settings.

1. Click **Setup ➤ Security Controls ➤ Field Accessibility**, and then select the Candidate object.

Notice that with the Field Accessibility tool, we start off by choosing the relevant object, rather than a user profile. Then, we can choose to set field-accessibility either by selecting a single field and then seeing its security settings for every profile (**View by Fields**), or by selecting a profile and then seeing security settings for every field (**View by Profile**).

2. Click **View by Fields**, and then select SSN from the `Field` drop-down list.

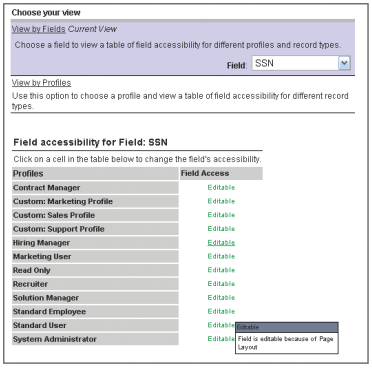


Figure 54: Field Accessibility Page

The table that appears shows us the accessibility settings for social security number for each profile. If you move your mouse over the value for `Field Access`, hover text indicates whether the security setting is the result of a page layout setting or a field-level security setting.

Let's first set our field-level security for hiring managers.

3. Next to the Hiring Manager profile, click **Editable**.

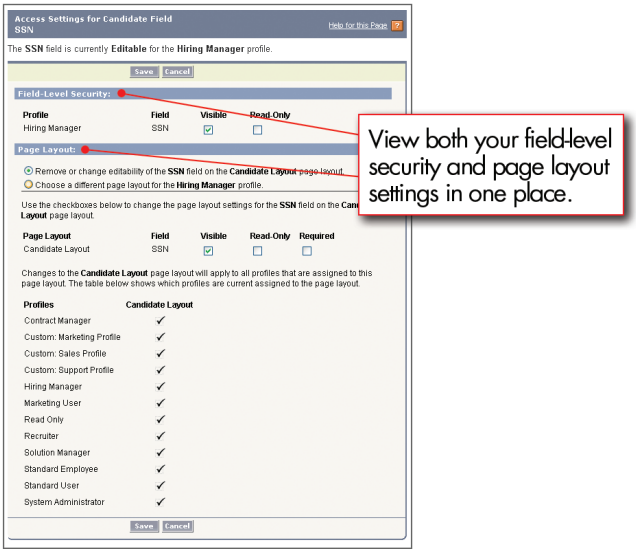


Figure 55: Access Settings Page

By clicking the value in a profile's `Field Access` column, we finally get to see the real reason why the Field Accessibility page is so convenient: we can view and edit a field's field-level security and page layout settings in the same page.

The Field-Level Security area includes the same checkboxes that we discussed previously—to hide the social security number from hiring managers, we can simply deselect the `Visible` checkbox as we did with the `Min Pay` and `Max Pay` fields on Positions.

Before we do that, however, take a look at the Page Layout area. Here you can either change the visibility and editability of the field on the page layout, or you can choose a different page layout for the Hiring Manager profile altogether. Since we are focused on the security of the `SSN` field in this exercise we don't need to touch any of the Page Layout area settings. However, you should remember that this tool is here for the future, especially if you are ever trying to figure out why a field is or is not visible to a particular user.

4. In the Field-Level Security area, deselect `Visible`.
5. Click **Save**.

To finish up, all we need to do is repeat the process for the Standard Employee profile.

6. Next to the Standard Employee profile, click **Editable**.
7. In the Field-Level Security area, deselect `Visible`.
8. Click **Save**.

All done! We've just finished the second piece of our security and sharing puzzle by defining field-level security for the sensitive fields in our Recruiting app. Now, for the final (and most complicated) piece of the puzzle, we need to specify the individual records to which each user needs access. We need to protect our data without compromising any employee's ability to perform his or her job.

Controlling Access to Records

By setting object- and field-level access rules for each of our three user profiles, we have effectively defined all of the objects and fields that any one of our Recruiting app users can access. In this section, we'll focus on setting permissions for the actual records themselves—should our users have open access to every record, or just a subset? If it's a subset, what rules should determine whether the user can access them? We'll use a variety of platform security and sharing tools to address these questions and make sure we get it right.

Introducing Organization-Wide Defaults

When dealing with record-level access settings, the first thing we need to do is to determine the organization-wide defaults (commonly called "org-wide defaults") for each object in our Recruiting app. Also called a sharing model, org-wide defaults specify the baseline level of access that the most restricted user should have. We'll use org-wide defaults to lock down our data to this most restrictive level, and then we'll use our other record-level security and sharing tools (role hierarchies, sharing rules, and manual sharing) to open up the data to other users who need to access it.

Org-Wide Defaults in Our Recruiting App

To determine the org-wide defaults that we'll need in our Recruiting app, we need to answer the following questions for each object:

1. Who is the most restricted user of this object?
2. Is there ever going to be an instance of this object that this user shouldn't be allowed to see?
3. Is there ever going to be an instance of this object that this user shouldn't be allowed to edit?

Based on our answers to these questions, we can determine the sharing model that we need for that object as illustrated in the following diagram.

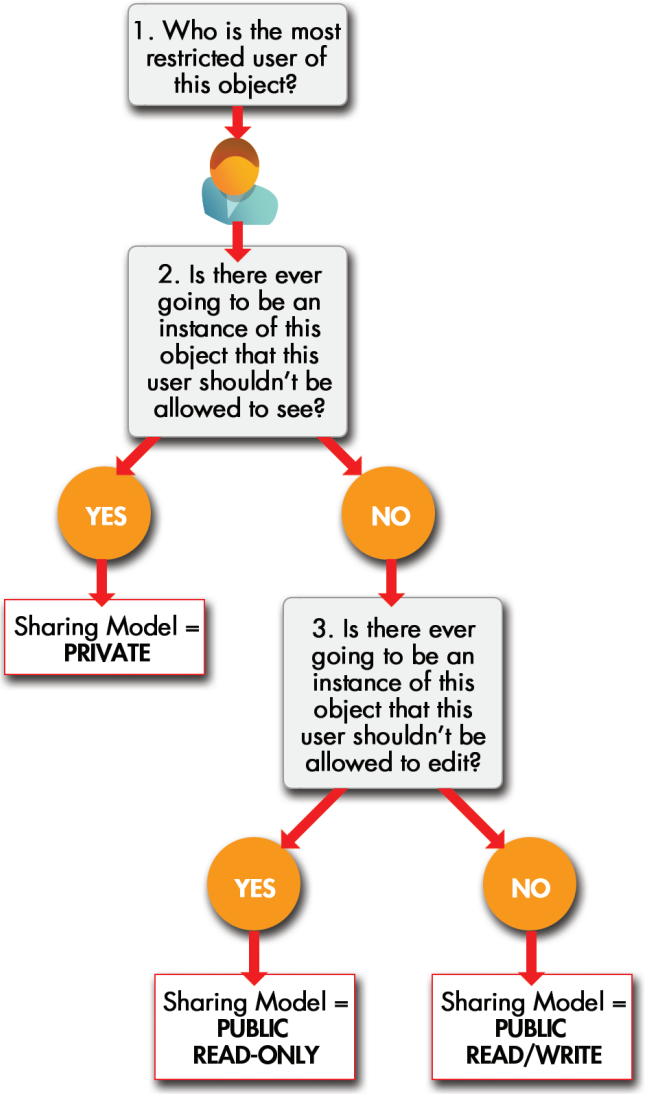


Figure 56: Determining the Sharing Model for an Object

For example, let's consider the Position object in our recruiting app. To refresh our memories, here's our table of required permissions:

Table 28: Revised Summary of Required Permissions

	Position	Candidate	Job Application	Review
Recruiter	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit	• Read • Create • Edit
Hiring Manager	• Read • Create • Edit*	• Read* (No SSN)	• Read* • Edit*	• Read • Create • Edit
Standard Employee	• Read (No min/max pay)	• Read* (No SSN)	• Read*	• Read** • Create • Edit**

* Only for those records that are associated with a position to which the hiring manager/standard employee has been assigned

** Only for those records that the standard employee owns

Now let's go through and answer our list of questions for Position:

Table 29: Determining Org-Wide Defaults for the Standard Employee Profile

Question	Answer
1. Who is the most restricted user of this object?	A member of the Standard Employee profile. All that they're allowed to do is view a position.
2. Is there ever going to be an instance of this object that this user shouldn't be allowed to see?	No. Although the values for the minimum and maximum pay are hidden from standard employees, they're still allowed to view all Position records.
3. Is there ever going to be an instance of this object that this user shouldn't be allowed to edit?	Yes. Standard employees aren't allowed to edit any Position record.

According to our flowchart, answering "Yes" to question #3 means that the sharing model for Position should be set to Public Read-Only.

Going through the rest of our recruiting objects required permissions, we can easily figure out their sharing models, too. The Standard Employee profile is the most restricted user for each object, and there are going to be Candidate, Job Application, and Review records that particular employees won't be able to view. Consequently, the sharing model for Candidates, Job Applications, and Reviews should all be set to Private.

Try It Out: Setting Org-Wide Defaults

Now that we've figured out the org-wide defaults for each of our recruiting objects, let's go ahead and implement them in our Recruiting app.

1. Click **Setup ➤ Security Controls ➤ Sharing Settings**. If you see an introductory splash page, click **Set Up Sharing** at the bottom of the page to skip to the actual tool.

The Sharing Settings page is where we control both org-wide defaults and sharing rules. We'll talk more about this page when we talk about sharing rules a little further down. For now, let's just edit our org-wide default settings.

2. In the Organization Wide Defaults area, click **Edit**.

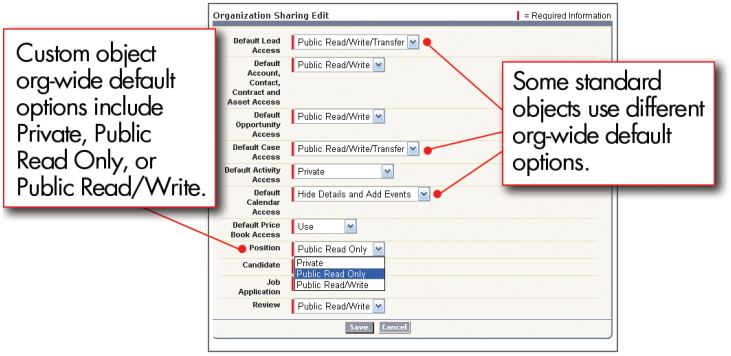


Figure 57: Org-Wide Defaults Edit Page

This page controls the org-wide defaults for every object in our organization. You'll notice that some standard objects (like leads and calendars) use a different set of org-wide default values than we have available for our custom recruiting objects. You can learn more about them in the online help. For now, let's just set our recruiting objects to the org-wide defaults that we decided on in the last section.

3. Next to `Position`, select Public Read Only.
4. Next to `Candidate`, `Job Application`, and `Review`, select Private.
5. Click **Save**.

Easy! Now that we've locked down our data with org-wide defaults, users are currently allowed to work on only Candidate, Job Application, and Review records that they own and allowed to view Position records that anyone owns. Because those settings are way too restrictive for any user to get any benefit out of our app, now we need to use role hierarchies, sharing rules, and manual sharing to open up Candidate, Job Application, and Review record access to those employees who'll need it.

Introducing Role Hierarchies

The first way that we can share access to records is by defining a role hierarchy. Similar to an org chart, a role hierarchy represents a level of data access that a user or group of users needs. Users assigned to roles near the top of the hierarchy (normally the CEO, executives, and other management) get to access the data of all the users who fall directly below them in the hierarchy. The role hierarchy ensures that a manager will always have access to the same data as his or her employees, regardless of the org-wide default settings. Role hierarchies also helpfully define groups of users who tend to need access to the same types of records—we'll use these groups later when we talk about sharing rules.

To illustrate, let's take a look at a portion of the role hierarchy for Universal Containers:

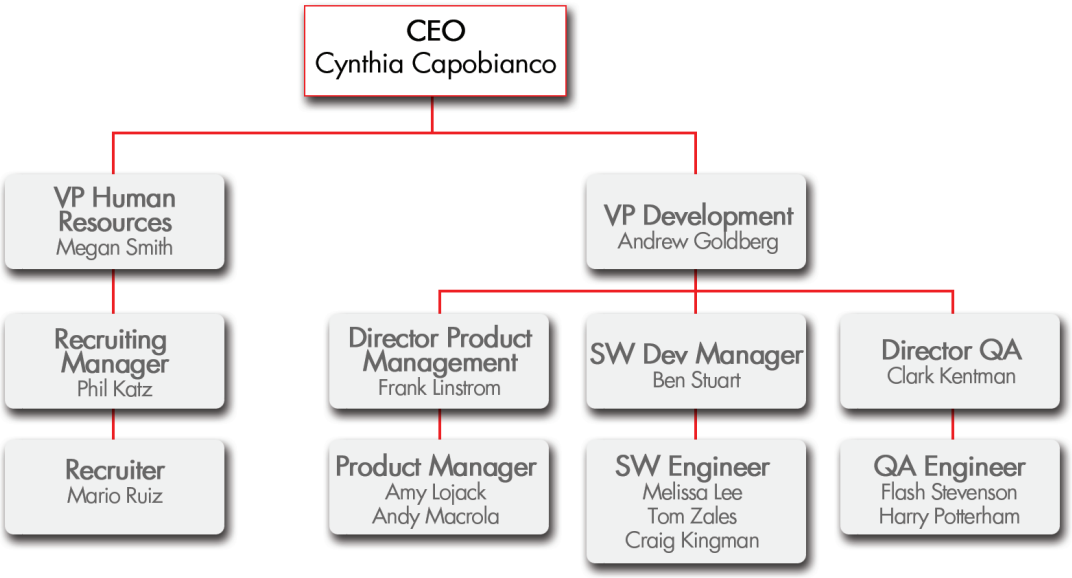


Figure 58: Universal Containers Role Hierarchy

Role hierarchies don't necessarily need to match your org chart exactly. Instead, each role in the hierarchy should just represent a level of data access that a user or group of users needs. For example, suppose your organization employs a corporate lawyer who needs to access all of the records in the app. One easy way to accomplish this is by assigning the lawyer to the CEO role in your organization's role hierarchy. Since the CEO role is placed at the top of the hierarchy, anyone assigned to that role automatically gets full access to any record in the organization. It doesn't matter that technically the lawyer appears below the CEO in the regular org chart.

Comparing Roles and Profiles

Although it's easy to confuse profiles with roles, they actually control two very different things.

As we learned earlier in this chapter, profiles control a user's object- and field-level access permissions. Indeed, a user can't be defined without being assigned to a particular profile, since the profiles specifies the apps and tabs that appear when he or she logs in, among a number of other useful things.

Roles, on the other hand, primarily control a user's record-level access permissions through role hierarchy and sharing rules. Although a role assignment isn't exactly required when we define a user, it would be foolish of us *not* to assign a role since it makes it so much easier to define our record-level permissions. Indeed, trying to define record-level permissions without

assigning a role to a user would be a lot like trying to travel from New York to San Francisco by car when there's an airplane available—there's just a much more efficient way of doing it!

Because profiles control object- and field-level access whereas roles influence record-level access, a user is typically assigned to one of each. To help you remember which controls what, remember: **R**oles control **R**ecords.

Role Hierarchies in Our Recruiting App

Given the Universal Containers role hierarchy that's pictured in *Figure 58: Universal Containers Role Hierarchy*, let's think about how implementing this hierarchy will open up certain kinds of record-level permissions to various users of our Recruiting app. Remember, since defining our org-wide defaults, our users are currently allowed to only view all Position records and to view and update other recruiting records that they own. That doesn't make our app all that useful. However, once we implement our role hierarchy, we'll automatically grant several kinds of record-level permissions to various users. For example:

- The CEO, Cynthia Capobianco, will be able to view and update every record that anyone else in the organization can view and update.
- The VP of Development, Andrew Goldberg, will be able to view and update any record that his managers or his managers' employees can view or update.
- The VP of Human Resources, Megan Smith, will be able to view and update any record that Phil Katz, her recruiting manager or Mario Ruiz, Phil's recruiter, can view and update.
- The Recruiting Manager, Phil Katz, will be able to view and update any record that is owned by Mario Ruiz, his recruiter.
- The Software Development manager, Ben Stuart, will be able to view and update any record that is owned by Melissa Lee, Tom Zales, or Craig Kingman, his software engineers.
- The director of QA, Clark Kentman, will be able to view and update any record that is owned by Flash Stevenson or Harry Potterham, his QA Engineers.
- The director of Product Management, Frank Linstrom, will be able to view and update any record that is owned by Amy Lojack or Andy Macrola, his product managers.

As we can see, the role hierarchy is very powerful in opening up data for people high up in the role hierarchy tree! However, let's look at some of the gaps that we still have in our record-level permissions:

- Megan Smith (and her whole recruiting team) won't be able to view any reviews that are owned by members of Andrew Goldberg's Development team, because she doesn't have a direct line down to any Development roles in the role hierarchy.
- Ben Stuart, the software development manager, also won't be able to see any reviews that were written by members of the QA or Product Management groups, even if QA engineers

or product managers interviewed candidates for a software engineering position in his group.

- Melissa Lee, a software engineer, won't be able to see the records for candidates that she's supposed to interview.

Clearly we'll need to use other record-level sharing methods to open up data between peers in the same group, and also between groups that appear in different branches of the role hierarchy (we'll get to those later in this chapter). However, the role hierarchy does give us a good start toward opening up record access, so let's take a look now at how to define it.

Try It Out: Defining a Role Hierarchy

Implementing a role hierarchy in the platform is easy once you have an idea of what the hierarchy should look like. It's best to just start with your company's org chart and then consolidate different job titles into single roles wherever possible. For example, if Ben Stuart's software development group has a staff software engineer and a junior software engineer, these positions can be consolidated into a single Software Engineer role in the hierarchy.

Once that's all squared away, we can get started actually defining the role hierarchy itself. For our exercise, we'll go ahead and use the role hierarchy that we talked about in the previous sections.

1. Click **Setup ➤ Manage Users ➤ Roles**. If you see an introductory splash page called Understanding Roles, click **Set Up Roles** at the bottom of the page to skip to the actual tool.

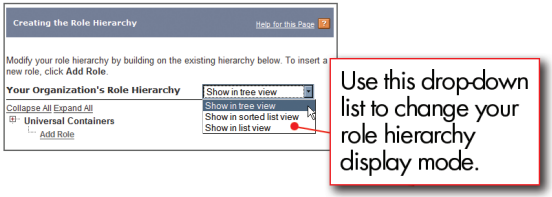


Figure 59: Empty Role Hierarchy Page in Tree View Mode

The default view for this page is the tree view, as indicated in the drop-down list on the far right side of the Role Hierarchy title bar. When creating a role hierarchy, it's probably easiest to stick with this or the list view, because they both make it easy to see how the roles all fit together in the hierarchy. The sorted list view is best if you know the name of a role that you want to find but aren't sure where it fits in the hierarchy (or are too lazy to click open all the tree nodes or scroll down a long list). For our purposes, we'll stick with the tree view for now.

When you first start defining a role hierarchy, the tree view displays a single placeholder node with the name of your organization. From this point, we need to add the name of the role that is highest up in the hierarchy—in our case, the CEO.

Note: If you're building your Recruiting app with a free Developer Edition organization, you may have a role hierarchy predefined as a sample. That's alright. You can still follow along and create some more roles.

2. Just under Universal Containers, click **Add Role**.
3. In the `Role Name` text box, enter CEO.
4. In the `This role reports to` text box, click the lookup icon and click **Select** next to Universal Containers.

By choosing the name of the organization in the `This role reports to` text box, we're indicating that the CEO role is a top-level position in our role hierarchy and doesn't report to anyone.

5. In the `Role Name as displayed on reports` text box, enter CEO. This text is used in reports to indicate the name of a role. Since you may not want a long role name, like Vice President of Product Development, taking up too much space in your report columns, it's advisable to use a shortened, yet easily identifiable, abbreviation.
6. Leave any other options, such as `Opportunity Access`, set to their defaults. These access options don't have anything to do with our Recruiting app, and only appear if you have the org-wide defaults for a standard object set to a level more restrictive than Public Read/Write.)
7. Click **Save**.

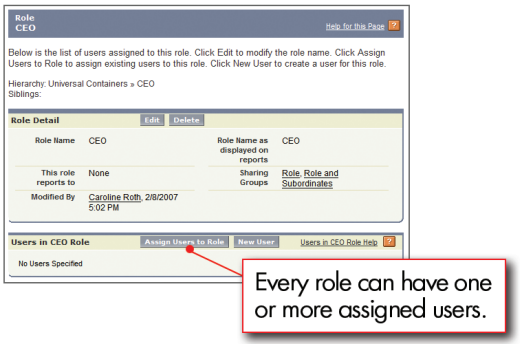


Figure 60: CEO Role Detail Page

Now that we've created our first role, we can assign the appropriate user to it.

8. In the CEO role detail page, click **Assign Users to Role**.
9. In the `Available Users` drop-down list, select All Unassigned.
10. Choose a user from the list (in our case, Cynthia Capobianco), and click **Add** to move her to the `Selected Users for CEO` list.
11. Click **Save**.

If we return to the main Roles page by clicking **Setup ➤ Manage Users ➤ Roles**, we can now see our new CEO role in the hierarchy. Defining the rest of the roles is just an exercise that you can do on your own according to *Figure 58: Universal Containers Role Hierarchy* on page 137. (If you don't define the role hierarchy, some of the tests that we talk about later won't work as described.)

Note: There's no need to assign users to every role at this point—we'll do that later when we test out our app.

Tip: To speed up the process of adding a new role, click **Add Role** directly under the name of the role to which the new role should report. When you do this, the `This role reports to` text box is automatically filled in with the name of the appropriate role.

Not too hard, right? With org-wide defaults and a role hierarchy in place, we're actually pretty close to finishing up our record-level access permissions. All we have left to do is share recruiting-related records between groups that appear in separate branches of the role hierarchy, and between peers in a single group. Fortunately, we can accomplish both of those tasks with a combination of sharing rules and manual sharing. We just need to figure out what's left that needs to be shared, and with whom.

What's Left to be Shared?

So what *is* left to be shared? After reviewing our table of required permissions, it turns out it's just a few more things (remember, since users always have access to the records that they own, we need to worry only about the read and update permissions for our record-level access settings):

- Recruiters need read and update access on every Position, Candidate, Job Application, and Review record that exists in the app.
- Hiring managers need:
  - Read and update access on Position and Job Application records on which they're the hiring manager
  - Read access on Candidate records for which they're the hiring manager

    - Read and update access on every Review record
- Interviewers need read access on the Candidate and Job Application records for people they're interviewing.

That shouldn't be too hard! Let's go do it.

Introducing Sharing Rules

First let's see what we can do with sharing rules. Sharing rules let us make automatic exceptions to org-wide defaults for particular groups of users. We've already defined several specific groups with the roles that we created in the previous section, but we can also make up other groups as needed.

The thing to remember with sharing rules is that, like role hierarchies, we can use them only to open up record access to more users. Sharing rules and role hierarchies can never be stricter than our org-wide default settings.

Sharing Rules in Our Recruiting App

Sharing rules work best when they're defined for a particular group of users that we can determine or predict in advance, rather than a set of users that is frequently changing. For example, in our Recruiting app, we need to share every position, candidate, job application, and review with every recruiter. Since recruiters all belong to either the Recruiting Manager or Recruiter roles in the role hierarchy, we can easily use a sharing rule to share those objects with the Recruiting Manager role and its subordinates.

Alternatively, consider another use case from our Recruiting app: interviewers need read access on the candidates and job applications for people they're interviewing. In this case, the set of interviewers is a lot harder to predict in advance—hiring managers might use different sets of interviewers depending on the position for which they're hiring, and the interviewers might come from different groups in the role hierarchy. As a result, this use case probably shouldn't be handled with sharing rules—the team of interviewers for any given manager is just too hard to predict.

Let's go through the set of required permissions we still need to implement and pick out the ones that would work best with sharing rules:

Use Case	Should we use a sharing rule?
Recruiters need read and update access on every Position, Candidate, Job Application, and Review record that exists in the app.	**Yes**. As we discussed previously, it's easy to pick out the group of recruiters in our role hierarchy.
Hiring managers need read and update access on Position and Job Application records on which they're the hiring manager.	**No**. It's too hard to predict which positions will be assigned to which hiring manager. We'll need to handle this use case some other way.
Hiring managers need read access on Candidate records on which they're the hiring manager.	**No**. Again, it's too hard to predict which positions will be assigned to which hiring manager.
Hiring managers need read and update access on every Review record.	**Yes**. Since we're not restricting which reviews a hiring manager gets to read and update, we can easily pick out all of the hiring managers from our role hierarchy and define a sharing rule for them.
Interviewers need read access on the Candidate and Job Application records for people they're interviewing.	**No**. As we discussed previously, it's hard to predict who will be a member of an interview team for a particular position.

Great! Now that we know the required permissions we want to implement with sharing rules, let's go ahead and define them.

Try It Out: Defining a Public Group for Reviewers

Before we dive head first into creating our sharing rules, we need to make sure that we have the appropriate public groups set up. A *public group* is a collection of individual users, other groups, individual roles, and/or roles with their subordinates. Using a public group when defining a sharing rule makes the rule easier to create and, more important, easier to understand later, especially if it's one of many sharing rules that you're trying to maintain in a large organization. You'll need to create a public group if you ever want to define a sharing rule that encompasses more than one or two groups or roles, or any individual.

Looking at the required permissions that we want to implement, there's just one object that needs a public group for its sharing rule: Reviews. Since both recruiters and hiring managers need read and update access, let's go ahead and make a public group called Reviewers that encompasses their roles in the role hierarchy.

1. Click **Setup ➤ Manage Users ➤ Public Groups**.
2. Click **New**.

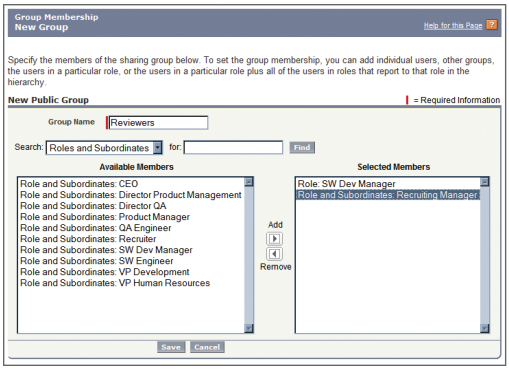


Figure 61: New Public Group Page

The New Public Group page allows you to choose other public groups, individual roles, individual roles including the roles' subordinates, or individual users.

3. In the `Group Name` text box, enter Reviewers.
4. In the `Search` drop-down list, choose Roles.
5. In the `Available Members` list, select Role: SW Development Manager, and click **Add**.
6. In the `Search` drop-down list, choose Role and Subordinates.
7. In the `Available Members` list, select Role and Subordinates: Recruiting Manager, and click **Add**.
8. Click **Save**.

Easy! Now we're ready to define our sharing rules.

Try It Out: Defining a Sharing Rule for Review Records

Since we just defined our Reviewers public group, let's go use it to define our sharing rule for Review records.

1. Click **Setup ➤ Security Controls ➤ Sharing Settings**.

Remember this page? We were last here when we defined our org-wide defaults.

2. In the `Manage sharing settings for` drop-down list, choose Review.

Choosing an object in this drop-down list allows us to focus in on the org-wide defaults and sharing rules for a single object at a time rather than looking at all of them in a long page—a really useful thing if you've got a large organization with several custom objects.

3. In the Sharing Rules area, click **New**.
4. In the `Review: owned by members of` drop-down list, select Public Groups.
5. Next to that drop-down list, choose Entire Organization.

Just as we talked about already, you can define a sharing rule only for a single public group, role, or role with all of its subordinates. By default, the platform includes a default public group that encompasses every user in your organization.

6. In the `Share with` drop-down list, select Public Groups.
7. Next to that drop-down list, choose Reviewers.
8. In the `Access Level` drop-down list, select Read/Write.
9. Click **Save**.
10. Click **OK** in the dialog box that says this operation could take significant time.

And that's it! We've just created a rule that shares reviews written and owned by any member of the organization with all recruiters and hiring managers. Since reviewers and hiring managers all need the power to read and update reviews, we handled everyone with a single sharing rule and a public group.

To finish up here, go ahead and create two more sharing rules according to the following table:

Table 31: Additional Sharing Rules

These records...	Owned by...	Should be shared with...	Access Level
Candidate	The role and subordinates of the Recruiting Manager	The role and subordinates of the Recruiting Manager	Read/Write
Job Application	The role and subordinates of the Recruiting Manager	The role and subordinates of the Recruiting Manager	Read/Write

Introducing Manual Sharing

Now let's talk about what we have left to do to define our sharing model. After implementing our sharing rules, the following required permissions remain:

- *Hiring managers need read and update access on Position and Job application records on which they're the hiring manager.*
- *Hiring managers need read access on Candidate records on which they're the hiring manager.*
- *Interviewers need read access on the Candidate and Job Application records for people they're interviewing.*

We didn't implement those required permissions with sharing rules because it was too hard for us to come up with a consistent group of users who would need access to a particular set of records. Really, this is where the job of the recruiter comes into play. A recruiter like Mario Ruiz owns the Position, Candidate, and Job Application records for jobs that he's trying to fill, and he also knows the hiring manager and interviewers who should be assigned to them.

Fortunately, we have one final type of record-access setting that allows Mario to share specific records with other specific users: manual sharing. With manual sharing, Mario can grant read or read/write access on records that he owns to any other user, role, or public group. Although it isn't automated like org-wide defaults, role hierarchies, or sharing rules, manual sharing gives Mario the flexibility to share particular records with the ever-changing groups of interviewers and hiring managers with whom he has to deal every day.

Try It Out: Defining a Manual Sharing Rule

Let's pretend that we're a recruiter like Mario and we need to share a particular Candidate record that we own with another role, group, or user:

1. On the detail page for the candidate, click **Sharing**.

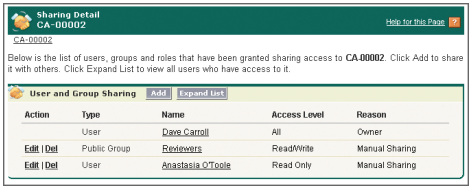


Figure 62: Sharing Detail Page

Since we own this Candidate record, we get to see details about who else can see the record and why. If we didn't own this record, there would be a message about not having sufficient privileges.

Tip: If we wanted to view the names of each user who has access to the record rather than just the names of the roles and public groups, we could click **Expand List** in this page. Although the operation can take some time depending on the number of users

in our organization, it's helpful to determine whether we need to define a manual sharing rule for a particular user or if he or she already has access.

2. Click **Add**.
3. In the `Search` drop-down list, choose whether we want to manually share the record with a user, public group, role, or role and subordinates.
4. In the `Currently Not Shared` list, select the user, public group, or role that should have access to the record, and click **Add**.
5. In the `Access Level` drop-down list, specify whether the user, public group, or role should have read or read/write access to the record.
6. Click **Save**.

Not too hard! When we roll out our Recruiting app to users, we'll have to train our recruiters to take these steps for the Position, Candidate, and Job Application records that their hiring managers and interviewers need to access. Once this training is complete, we will have implemented all of the required sharing and security settings that we discussed at the beginning of the chapter—well done!

Putting It All Together

Congratulations! We've just implemented all of our required security and sharing settings, first by defining object-level access with profiles, then by securing field-level access with field-level security, and finally by defining record-level access using org-wide defaults, role hierarchies, sharing rules, and manual sharing.

We learned about the difference between object-, field-, and record-level security, and how profiles and roles work together to first determine the objects and tabs that a user can possibly use, and then the specific records that the user can actually access and edit.

Let's now try it out for ourselves. To do so, we'll first have to define a number of users, and then we can play around with creating records and seeing who has access to what.

Try It Out: Creating Users for Our Recruiting App

To really put our Recruiting app through its paces, we'll first need to define the following users and assign a couple of them to some of the recruiting records that we imported earlier.

Note: If you're implementing the Recruiting app in a Developer Edition organization, you'll have only one additional user to play with besides the System Administrator

user. You can still try out all of the use cases that we describe here, but you'll have to update the user's profile and role for whatever use case you're working on.

Table 32: Recruiting App Users

User's First and Last Name	Title	Profile	Role	Owner of
Phil Katz	Recruiting Manager	Recruiter	Recruiting Manager	• DBA position • Mary Jane, candidate for DBA • Job Application linking the DBA position with Mary Jane
Mario Ruiz	Senior Recruiter	Recruiter	Recruiter	• Documentation Writer position • George Schnell, candidate for Documentation Writer • Job Application linking the Documentation Writer position with George Schnell
Ben Stuart	Software Development Manager	Hiring Manager	SW Dev Manager	

User's First and Last Name	Title	Profile	Role	Owner of
Melissa Lee	Staff Software Engineer	Standard Employee	SW Engineer	
Amy Lojack	Senior Product Manager	Standard Employee	Product Manager	• Review for the job application associated with the DBA position*

* You'll need to create this review record because it isn't part of the sample import data.

Let's walk through the creation of Phil Katz. Then you can finish the other four users on your own:

1. Click **Setup ➤ Manage Users ➤ Users**.
2. Click **New User**.
3. Fill out the fields in the User Edit page according to *Recruiting App Users*.

Because we're defining this user to test the security and sharing settings of our app, enter a real email address that you have access to in the `Email` field and then a "fake" email address in the `Username` field (for example, phil.katz@recruiting.com). We'll use the "fake" value in the `Username` field to log in to the app as Phil Katz, but we'll get Phil's automatically generated password at the real email account that you specified. Without that password, we'd never be able to log in! (For a real user, both `Email` and `Username` should generally have the same value.)

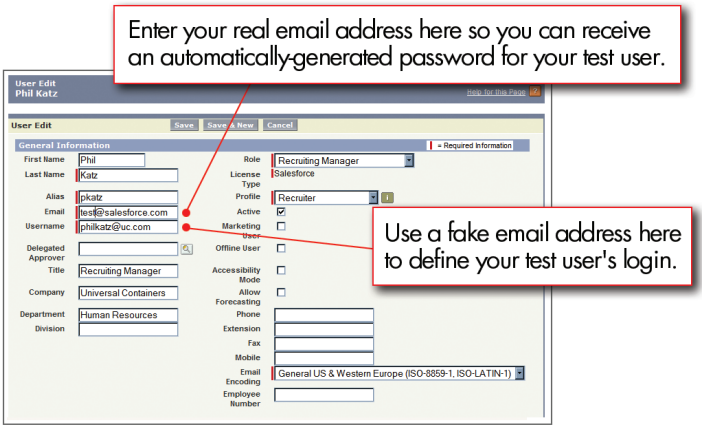


Figure 63: New User Edit Page

4. Click **Save**.

Now that we've created the Phil Katz user, let's give him ownership of the DBA position and its associated Job Application and Candidate records.

5. Click the Positions tab.
6. From the `View` drop-down list, select All and click **Go**.

Tip: If you want to see more than just the `Position Title` field in this view, click **Edit** next to the `View` drop-down list and add additional fields in the Select Columns section.

7. Click **DBA**.
8. Next to the `Owner` field, click **Change**.
9. Click the lookup icon and choose Phil Katz.
10. Click **Save**.
11. In the Job Applications related list, click the name of the listed Job Application and repeat Steps 8-10.
12. Click the ID of the associated Candidate on the Job Application detail page and repeat Steps 8-10.

All done! Now create the other four users in *Recruiting App Users* on page 148 and assign them ownership of the indicated records. To fully complete this, note that you'll have to create Amy Lojack's review.

Try It Out: Verifying that Everything Works

Now that we've got data assigned to actual users, let's go through our Recruiting app and see how the security and sharing permissions that we defined in this chapter play out:

1. First log in as Mario Ruiz—verify that he can see and edit both positions, all three candidates, all three job applications, and Amy Lojack's review. Verify that the **New** buttons are there for all recruiting objects.
2. Log in as Melissa Lee—verify that she can view positions but that there's no **New** button. Verify that she can't see any candidates, reviews, or job applications.
3. Log in as Ben Stuart—verify that he can view positions and that there's a **New** button. Verify that he can't see any candidates or job applications. Verify that he can view reviews and that there's a **New** button. (What do reviews look like? Can he see the names of the candidates and job applications on them?)
4. Log in again as Mario Ruiz—have him manually share read/write access on the Jr. Software Engineer position with Ben. Have him manually share read access on the candidate with Melissa and Ben. Have him manually share read access on the job application with Melissa and read/write access on the job position with Ben.
5. Log in again as Melissa Lee—verify that she can now see the candidate and job application that Mario just shared with her but that she can't see the candidate's social security number. Have her create a review for that candidate.
6. Log in again as Ben Stuart—verify that he can edit the Jr. Software Engineer position. Verify that he can read and update Melissa's review. Verify that he can update the job application to suggest that they hire the candidate.

How did we do? If all of these use cases worked correctly, you've just successfully set up security and sharing for our Recruiting app! Now let's go incorporate some business logic into our app with workflow rules.

Chapter 8

Using Custom Workflow and Approval Processes

In this chapter ...

Now that we've set up all our custom objects and made sure they're secure, it's time to start looking at ways we can make our Recruiting app more powerful.

Up to this point, we've created little more than a glorified database—it holds all of the information we need and allows us to search for records based on various criteria, but it doesn't help our recruiters and hiring managers perform the functions of their jobs more effectively. There's no automation that lets a recruiter know when the status of a candidate has changed or when a new position has been entered into the system.

Indeed, when any changes are made, users have to remember to notify one another of the change or else rely on the fact that others will find the updates on their own. Neither solution is practical for the long term and invites the possibility that the Recruiting app won't be adopted consistently by all the employees at Universal Containers. How can we build processes into our app so that users won't need to rely on manual methods of communication to inform others of changes?

Once again, the Force.com platform helps us out with more built-in functionality. In this case, the tools that we'll use to solve our process automation problem are called *workflow* and *approval processes*. We'll take a look at the various ways that we can use workflow first, and then we'll tackle an approval process at the end of the chapter.

Introducing Workflow

Workflow is a Force.com business-logic engine that allows us to automatically send email alerts, assign tasks, or update field values based on rules that we define. Any time that changes to a record meet the conditions in a workflow rule, the platform automatically performs any actions associated with the rule.

For example, let's say Ben Stuart, a software development manager, has decided to extend an offer to Ethan Tran, a bright young candidate who's interested in the Jr. Software Engineer position. At Universal Containers, it's a recruiter's job to extend offers, but in this case, Mario Ruiz, the recruiter responsible for the Jr. Software Engineer position, doesn't know whether or not Ben has made a decision unless Ben emails or calls him directly.

Instead of relying on Ben to remember to tell Mario, we can set up a simple workflow that triggers the assignment of the appropriate task whenever the status on a Job Application record is set to Extend an Offer or Rejected. As soon as Ben changes the status of the candidate's Job Application, workflow creates the appropriate task and sends Mario a notification email, as shown in the following diagram.

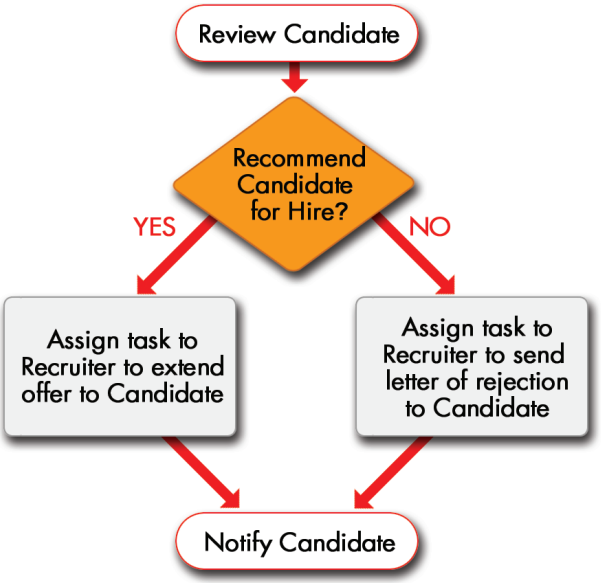


Figure 64: Automatically Assigning a Task to a Recruiter Using Workflow

Pretty powerful, isn't it? In general, if we can come up with a standard rule that specifies when a particular event should happen, we can make it happen automatically with workflow. Workflow is one of the secret ingredients that's going to transform our Recruiting app from a glorified database into a fully functional tool that everyone finds useful.

Now that we've got a general idea of what workflow's all about, let's take a closer look at the individual components that make up workflow: rules, tasks, field updates, and alerts.

Introducing Workflow Rules

In general, a *workflow rule* is the main container for a set of workflow instructions. It includes the criteria for when the workflow should be activated, as well as the particular tasks, alerts, and field updates that should take place when the criteria for that rule are met.

Every workflow rule must be based on a single object you choose when you define the rule. This object influences the fields that are available for setting workflow activation criteria.

For example, if we define a workflow rule for the Job Application object, we'll be able to set workflow activation criteria based on the values of fields like `Job Application Number` and `Status`. We can also set workflow activation criteria based on standard fields, like `Record Owner` or `Created Date`, as well as fields based on the currently active user when a rule is evaluated, such as their `Role` or `Time Zone`.

We'll look at all the ways that we can set workflow activation criteria when we get to building our own workflow rules a little later. For now, just understand that the platform makes it very easy to create detailed workflow rules that target specific situations.

Introducing Workflow Actions: Tasks, Field Updates, and Alerts

When a workflow rule is triggered, the following types of actions can occur:

Workflow Tasks

A *workflow task* assigns a task to a user according to a particular template. Just as in Microsoft Outlook, tasks include information about something that needs to be done by a certain time, such as making a telephone call or returning a library book. Assigned tasks appear in a user's My Tasks related list on their Home tab and generate reminder messages that pop up when a user logs in.

When we define a workflow task, we provide default values for the `Assignee`, `Subject`, `Status`, `Priority`, and `Due Date` fields for tasks that are generated by an associated workflow rule. We can also make sure that a notification email is sent to the assignee when a task is automatically generated.

Workflow Field Updates

A *workflow field update* changes the value of a particular field on the record that initially triggered the workflow rule.

Workflow Email Alerts

A *workflow email alert* sends an email according to a specified email template. Unlike workflow tasks, which can only be assigned to users of the app, workflow alerts can be sent to any user or contact, as long as they have a valid email address.

Note: A fourth type of action, a *workflow outbound message*, sends data to an external Web service, such as another on-demand application. Outbound messages are used primarily with composite apps, so we won't be using them in this chapter.

A workflow rule can include any combination of these actions when the rule is triggered. For example, one rule might send out an alert and update two fields on a particular record. The action that one workflow rule takes can also trigger the execution of another workflow rule.

Workflow in Our Recruiting App

Now that we've oriented ourselves to the different components involved with workflow, let's take a look at our Recruiting app and see how we can use workflow to help us build out the requirements that we talked about in *Chapter 2: About the Sample Recruiting App* on page 15. Then we'll spend time implementing the rules that we come up with here.

We've already talked about one instance where workflow will give us a big advantage: automatically assigning a task to a recruiter when the status of a Job Application record changes to Reject or Extend an Offer. This will be a great start, but what else can we do?

If we look back at our last chapter on security and sharing, recall that we wanted to grant both recruiters and hiring managers permission to create new positions, but that ultimately we always wanted a recruiter to own those records because filling them is the recruiter's job responsibility. We hinted in the security and sharing chapter that we could accomplish this with workflow, and indeed we can! We'll simply need to use a workflow field update to change the record owner of a Position record to a recruiter if it was originally created by a hiring manager. To prevent a single recruiter from getting overloaded with all these additional positions, we'll also use another platform feature, *queues*, to divvy up the orphaned position records fairly. We'll place the record in a queue of other position records without owners, and then let individual recruiters claim the positions they want.

For a third way of using workflow, let's think about how position availability is advertised throughout Universal Containers. Like many organizations, Universal Containers prefers to fill positions with employee referrals, but employees often aren't aware of which positions are currently open. We can use a workflow alert to automatically send email to every employee

whenever a new position opens up. This way employees learn about positions as they become available and can immediately think of friends and family members who might be interested.

Sound good? While there are probably many more ways to use workflow to build a killer on-demand Recruiting app, let's stick with these three for now since they'll give us a good example of each of the three types of workflow actions that are available. To summarize what we'll be building:

1. A workflow task that assigns a task to a recruiter when the status of a Job Application changes to Rejected or Extend an Offer
2. A workflow field update that reassigns ownership of a Position that's created by a hiring manager to a queue of Position records without owners, so that individual recruiters can claim ownership of the positions they want
3. An workflow alert that sends an email to every employee when a new Position is created

Now let's get started!

Creating Workflow Rules That Assign Tasks

For our first use of workflow, we want to create two rules: one that assigns a "Send Rejection Letter" task to the relevant recruiter whenever the `Status` field of a Job Application record is set to Rejected, and one that assigns an "Extend Offer" task to the relevant recruiter whenever the `Status` field of a Job Application record is set to "Extend an Offer." Let's start with the "Send Rejection Letter" workflow rule.

Try It Out: Creating the "Send Rejection Letter" Workflow Rule

To make this rule entirely functional, we'll need to define both a workflow rule, which specifies the criteria for when the rule should be executed, and a workflow task that includes the "Send Rejection Letter" task template. Although we can define workflow components in any order, let's start with defining the workflow rule itself. It'll save us a couple of clicks a little later when we define the rule's associated workflow task.

1. Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Workflow Rules**.
2. If you see an introductory splash page, simply click **Continue**.
3. Click **New Rule**.

The first thing we need to do is select the object that will be associated with our workflow rule. As we talked about earlier, every workflow rule must be associated with a single object to

determine the fields that we can use to set criteria. Since we need to trigger this workflow rule when the `Status` field of a Job Application record is set to "Reject," we need to select Job Application here:

4. In the `Select object` drop-down list, choose Job Application, and click **Next**.

Now it's time to set our workflow rule details:

5. In the `Rule Name` text box, enter Send Rejection Letter.
6. In the `Description` text box, enter "Send a rejection letter when a hiring manager changes the status of a job application to Rejected."

Now use the Evaluation Criteria area to specify when this rule should be evaluated. We can choose `When a record is created, or when a record is edited and did not previously meet the rule criteria` (which repeatedly triggers the rule every time a record is saved only until the rule's criteria are met), `Only when a record is created` (which ignores updates to existing records), or `Every time a record is created or edited` (which repeatedly triggers the rule every time the record is saved). Since we don't want to assign duplicate tasks to a recruiter every time the record is saved, we'll choose the first option.

7. Under "Evaluate rule," select `When a record is created, or when a record is edited and did not previously meet the rule criteria`.

To finish up defining the rule, we need to set the criteria that will trigger execution of the rule's associated actions. Every workflow rule requires at least one row of filter criteria, but we can set as many filters as we want using additional rows.

8. In the first row of Rule Criteria filters:
   - Set the `Field` column to Status.
   - Set the `Operator` column to equals.
   - Set the `Value` column to Rejected.
9. Click **Save & Next**.

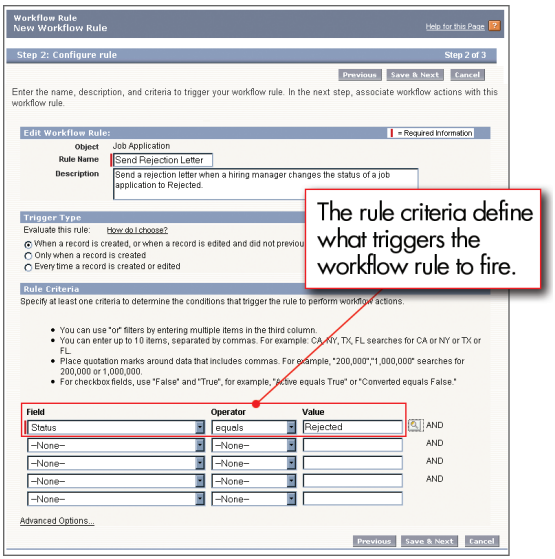


Figure 65: Creating a New Workflow Rule

At this point, we've just defined our "Send Rejection Letter" workflow rule. If we canceled out of the workflow wizard and clicked **Setup ➤ Customize ➤ Workflow & Approvals ➤ Workflow Rules**, we'd see it in the list view. However, because workflow rules aren't all that useful without an associated action, the workflow wizard takes us directly to a screen where we can define the "Send Rejection Letter" workflow task. Let's work through that now.

Try It Out: Creating the "Send Rejection Letter" Workflow Task

The Specify Workflow Actions step of the workflow wizard allows you to specify the workflow tasks, field updates, and alerts that should occur when the condition of our workflow rule is met.

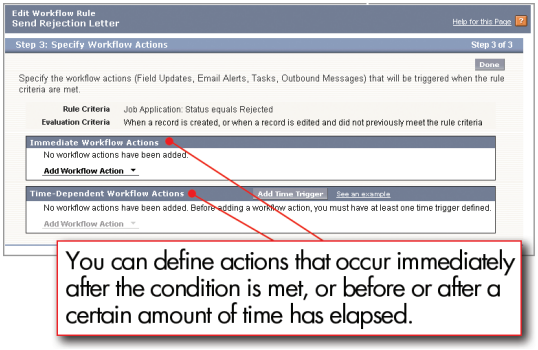


Figure 66: Specifying Workflow Actions

You can either define actions that occur immediately after the condition is met, or you can define actions that occur before or after a certain amount of time has elapsed (for example, seven days before the value of the `Hire By` field, or three days after the workflow rule is triggered). We'll talk more about these *time-dependent workflow actions* a little later. For now, we just need to define a single workflow task that executes as soon as our rule criteria is met:

1. In the Immediate Workflow Actions area , click **Add Workflow Action** and select **New Task**.

Did you notice that the `Object` field is already filled in with Job Application? Here's where we saved ourselves a couple of clicks—if we'd started building our workflow task before our workflow rule, we would have needed to specify the object with which the task would be associated. That's because, like workflow rules, workflow actions need to be associated with a single object.

In this case, we got to skip a page because the object with which a workflow action is associated must match the object of a workflow rule that uses it. For example, if we had a workflow rule associated with the Candidate object, any task, field update, or alert that's triggered by our Candidate workflow rule must also be associated with the Candidate object. Because we started building our "Send Rejection Letter" workflow task in the wizard right after defining our "Send Rejection Letter" workflow rule, the platform knew that the object associated with our task had to match the rule that we'd already built. That's why our new workflow task is already associated with the Job Application object.

The rest of the fields on this edit page make up the template for any "Send Rejection Letter" tasks that our workflow rule will generate.

2. Next to the `Assigned To` field, click the lookup icon ().

Here we can choose the assignee of the task either by specifying a particular user, role, or the owner of the Job Application record that triggered the workflow rule in the first place. Since recruiters always own the Job Application records of the positions that they're responsible for, and recruiters are responsible for sending rejection letters at Universal Containers, let's select the record owner:

3. In the `Type` drop-down list, choose Owner.
4. Click **Record Owner**.

Caution: If you thought that choosing the Recruiter role for the `Assigned To` field might have been another valid option, be careful. If the assignee of a workflow task is a role that contains more than one assigned user, the person who triggered the rule will become the task assignee instead. For this reason, you never want to assign workflow tasks to roles unless you're sure that only one user will ever be assigned to them at a time.

The rest of the workflow task fields are easy:

5. In the `Subject` text box, enter Send Rejection Letter.
6. In the `Due Date` drop-down lists, choose Rule Trigger Date plus 2 days.

This `Due Date` setting will give our recruiters two days to notify the candidate after they're first assigned the task.

7. In the `Status` drop-down list, choose Not Started.
8. In the `Priority` drop-down list, choose High.

The `Notify Assignee` checkbox allows us to send an email to the assignee as soon as the task is created by the workflow rule. This ensures that the assignee knows about the task without having to log in to the application on a regular basis, so let's select it.

9. Select the `Notify Assignee` checkbox.
10. Click **Save**.
11. Click **Done**.

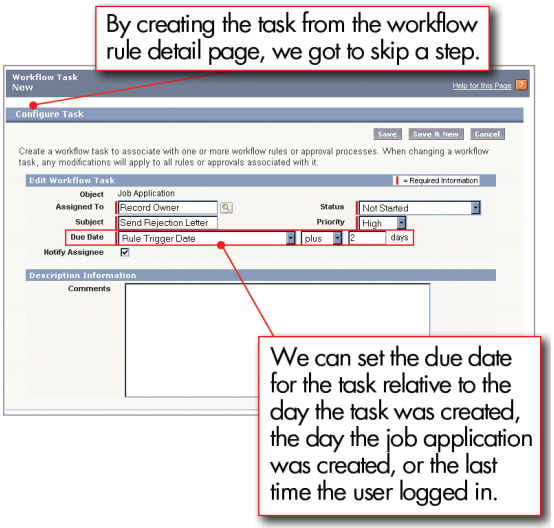


Figure 67: Creating the "Send Rejection Letter" Workflow Task

At this point, we finally get to see the detail page for the workflow rule that we just created. It includes the workflow rule criteria and a list of the associated actions. The only thing that remains to be done is to active the rule.

12. Click **Activate**.

All done! We've just created our first workflow rule and task. You'll find that the remaining workflow actions all operate in a very similar way, so we'll speed through the rest of them, focusing just on the fields and options that are unique to each. First, let's finish up our second workflow rule that assigns a task by quickly creating the "Extend an Offer" workflow rule.

Try It Out: Creating the "Extend an Offer" Workflow Rule and Task

To wrap up our first use case, we need to create another workflow rule for when the status of a Job Application is set to Extend an Offer. This rule and task are almost identical to the "Send Rejection Letter" workflow rule and task, so we'll just list the values that you'll need in the following two tables.

Table 33: Values for Creating the "Extend an Offer" Workflow Rule

Field	Value
Object	Job Application
Name	Extend an Offer

Field	Value
Description	Make an offer when a hiring manager changes the status of a Job Application to Extend Offer.
Evaluation Criteria	When a record is created, or when a record is edited and did not previously meet the rule criteria
Filter Criteria	Status equals Extend an Offer

Table 34: Values for Creating the "Extend an Offer" Workflow Task

Field	Value
Assigned To	Record Owner
Subject	Extend an Offer
Due Date	Rule Trigger Date plus 1 days
Status	Not Started
Priority	High
Notify Assignee?	Yes

All done! Make sure the "Extend an Offer" rule is also activated, and let's go try out one of our new workflow rules.

Look at What We've Done

Let's try out our new "Send Rejection Letter" workflow rule and see what happens:

1. Click the Job Applications tab, and select a Job Application record.
2. Click **Edit**, and change the `Status` field to Rejected.
3. Click **Save**.

The Send Rejection Letter task automatically appears in the Open Activities related list on the Job Application detail page, as shown in the following screenshot.

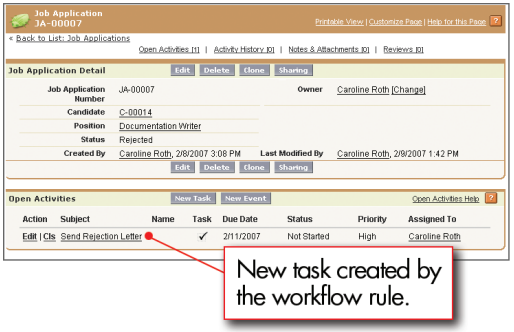


Figure 68: Open Activities Related List on Job Application Detail Page

Pretty neat, don't you think? Not only that, but if you check in the recruiter's email inbox, he should have received an automatically-generated email notification that looks like the following:

```
Caroline Roth has assigned a new task to you: Send Rejection Letter
Job Application: JA-00007 Due Date: 2/11/2007
For more details on this task, click on the link below:

https://na1.salesforce.com/00Tx04123s5k1
```

The link in the email message takes the recruiter directly to the task detail page, where he or she can locate the contact information of the candidate and update the task status after the task is completed. The task also shows up in the My Tasks area on the recruiter's Home tab and a reminder message will pop up in another day if the recruiter hasn't yet changed the task's `Status` to Completed. Suddenly, our Recruiting app has become a lot more interactive and powerful!

Creating a Workflow Rule That Updates Fields

For our next use case, we want to create a workflow rule that ensures the owner of a new Position record is always a recruiter. To do this, we're going to define a workflow field update that reassigns ownership of a position that's created by a hiring manager to a *queue* of position records without owners. Once a position record is in that queue, individual recruiters can claim ownership of the positions they want. But before we jump ahead of ourselves, let's stop a moment. Just what, exactly, is a queue?

Introducing Queues

Much like a collection of items in a lost and found drawer, a queue is a collection of records that don't have an owner. Users who have access to the queue can examine every record that's in it and claim ownership of the ones they want.

Queues are traditionally used in sales and support organizations to distribute new leads and support cases to the employees who have the most availability. Because the platform natively supports queues for Leads, Cases, and any custom object, we can create a queue for the Recruiting app's Position object.

Try It Out: Creating a Queue for Positions

To define a queue, we simply need to know the types of records that can be placed in the queue (in our case, Positions), and the users who are allowed to pull records out of it (in our case, Recruiters). Once we know those two things, defining the queue itself is just a matter of a few simple clicks:

1. Click **Setup ➤ Manage Users ➤ Queues**.
2. Click **New**.
3. In the `Queue Name` text box, enter Unclaimed Positions Queue.
4. In the `Queue Email` text box, enter an email address, such as recruiters@universalcontainers.com.
5. Select `Send Email to Members`.

Notice here that if recruiters@universalcontainers.com was a real email distribution list that went to all recruiters, we wouldn't need to select `Send Email to Members`. We do it here only because recruiters@universalcontainers.com is a fake email address and can't be used for testing later. The following table outlines the options you have for notifying queue members when new records are added to the queue:

Table 35: Options to Specify How Queue Members are Notified of New Records

	Queue Email Not Specified	Queue Email Specified
Send Email to Members Checkbox Not Selected	Because a queue email is not specified, individual queue members are always notified, regardless of the `Send Email to Members` checkbox.	Only the specified queue email address is notified.

	Queue Email Not Specified	Queue Email Specified
Send Email to Members Checkbox Selected	Because a queue email is not specified, individual queue members are always notified, regardless of the `Send Email to Members` checkbox.	The specified queue email address *and* individual queue members are notified. Note that if an individual queue member also receives emails sent to the specified queue email address, they'll receive duplicate notifications.

6. In the Supported Objects section, move Position into the `Selected Objects` list.

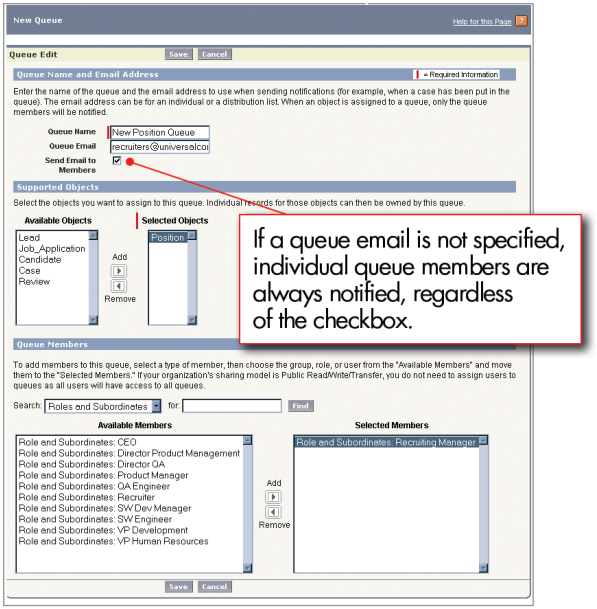


Figure 69: Defining a Queue

As you can see, a single queue can handle multiple objects—the platform allows you to do this so that you don't have to define multiple queues for the same group of users.

7. In the Queue Members section, select Roles and Subordinates from the `Search` drop-down list.
8. Move Role and Subordinates: Recruiting Manager to the `Selected Members` list.
9. Click **Save**.

Perfect! We've just defined a new queue that can act as a temporary owner for all of the Position records that are created by hiring managers. Whenever a position is placed in the queue, all recruiters are notified, and the appropriate person can claim ownership. All we need to do now is define the workflow rule that places those Position records into the queue.

Try It Out: Creating a Workflow Rule That Updates Fields

Now that we've got our queue ready to go, we can go ahead and define our workflow rule:

1. Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Workflow Rules**.
2. Click **New Rule**.
3. In the `Select object` drop-down list, choose Position, and click **Next**.
4. In the `Rule Name` text box, enter Assign Position to Recruiter.
5. In the `Description` text box, enter "Reassign position records to a recruiter if they were created by another type of employee."

While we know that recruiters should almost always own Position records, we don't want to impede the organization if there's a special case in which a non-recruiter should own the record instead. Let's choose to evaluate this rule only when a record is created so that if an exception needs to be made, the workflow won't supersede any changes that were made by a recruiter.

6. In the Evaluation Criteria area, select `Only when a record is created`.

Finally, we need to make sure that this rule is initiated whenever a Position record is created by someone who isn't a recruiter or a recruiting manager. We can specify this filter criteria in a single row by using a comma in the `Value` column as follows:

7. In the first row of filters:
   - Set the `Field` column to Current User: Role.
   - Set the `Operator` column to not equal to.
   - Set the `Value` column to Recruiter, Recruiting Manager.
8. Click **Save & Next**.

Now let's create the field update action for this workflow rule:

9. In the Immediate Workflow Actions area , click **Add Workflow Action**, and select **New Field Update**.
10. In the `Name` text box, enter Reassign Position to Queue.
11. In the `Description` text box, enter "Assign the Position to the Unclaimed Positions Queue."

12. In the `Field to Update` drop-down list, choose Owner.

Once you make a selection in this drop-down list, new options appear just below depending on the selection that you made.

13. In the `Owner` drop-down list, choose Queue.
14. Click the lookup icon (), and select Unclaimed Positions Queue.
15. Select `Notify Assignee`.
16. Click **Save**.

Not too hard! Before we leave this workflow rule behind, though, let's give it a second workflow action—one that ensures that no positions will languish in the queue without being claimed by a recruiter. This time, we'll use a time-dependent workflow action.

Introducing Time-Dependent Workflow Actions

As we mentioned earlier, time-dependent workflow actions are actions that occur before or after a certain amount of time has elapsed (for example, seven days before the value of the `Hire By` field, or three days after the workflow rule is triggered). We can use time-dependent workflow actions to fire tasks, field updates, and email alerts while the condition of a workflow rule remains true.

For example, the goal of reassigning Position records to the Unclaimed Positions queue is to have the appropriate recruiter take ownership. However, there might be situations in which a position is placed in the queue and no recruiter claims it. Rather than leaving the position to languish unclaimed in the queue, we can define a time-dependent workflow action that alerts the recruiting manager if no recruiter claims a Position record within a certain number of days. Because this action only takes place while the workflow condition remains true (that is, while the position is owned by a non-recruiter), the manager will only be alerted when necessary.

Neat, huh? Let's see how time-dependent actions are defined.

Try It Out: Creating the "Notify Recruiting Manager" Time-Dependent Workflow Task

At this point, we should still be on the Edit Workflow Rule page for our Assign Position to Recruiter Queue workflow rule. If not, you can return there by clicking **Setup ➤ Customize ➤ Workflow & Approvals ➤ Workflow Rules**, clicking Assign Position to Recruiter, and then clicking **Edit** in the Workflow Actions area.

Before we can define a time-dependent workflow task, we first must specify a *time trigger*. Time triggers define when the time-dependent workflow actions should fire.

1. Click **Add Time Trigger**.

In this case, we want our recruiting manager to be notified three days after a position has been assigned to the Unclaimed Positions queue.

2. Use the text box and drop-down lists to specify 3 Days After Rule Trigger Date.
3. Click **Save**.

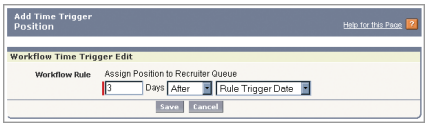


Figure 70: The Time Trigger Edit Page

Our time trigger is now listed in the Time-Dependent Workflow Actions area. The **Add Workflow Action** drop-down button is now active, and we can define our workflow task as usual.

4. In the Time-Dependent Workflow Actions area, click **Add Workflow Action** and select **New Task**.
5. In the `Assigned To` field, select the Recruiting Manager role.

Note: Remember, workflow tasks should only be assigned to a role if you're confident that only one user will ever be assigned to that role at a time. If the role contains multiple users, the owner of the workflow rule is assigned the task.

6. In the `Subject` field, enter Assign Unclaimed Position Record to Recruiter.
7. Set the `Due Date` field to Rule Trigger Date plus 4 days.

Since this workflow action won't fire until three days after the original Rule Trigger Date, making the `Due Date` four days after the Rule Trigger Date gives the recruiting manager one additional day to assign the position to a recruiter.

8. Set the `Status` field to Not Started.
9. Set the `Priority` field to High.
10. Select `Notify Assignee`.
11. Click **Save**.
12. Click **Done**.

Almost done! At this point, all we need to do is activate our workflow rule. However, if you click the **Activate** button now, an error message appears saying that the Default Workflow User must be set before activating the rule. What's that?

The Default Workflow User is the user who should be assigned as the owner of a workflow action if the user who originally triggered the rule is no longer active. This setting is required if you want to use time-dependent workflow actions. To set it:

13. Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Settings**.

Note: If you click **Activate** without previously setting the Default Workflow User and then click **OK** in the error dialog, you're sent directly to the Workflow & Approvals Settings page.

14. Set the `Default Workflow User` field to any user in your organization. In most cases, it's best to choose a system administrator.
15. Click **Save**.

Now we can activate the workflow rule:

Note: If you reached the Workflow & Approvals Settings page by clicking **Activate** and then **OK** in the error dialog, clicking **Save** also automatically activates your workflow rule.

16. Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Workflow Rules**.
17. Click **Activate** next to the Assign Position to Recruiter workflow rule.

Look At What We've Done

Great! We've now got a workflow rule that moves Position records created by non-recruiters to a queue of unclaimed positions, and if not claimed by a recruiter in three days, assigns a task to the recruiting manager.

To try it out, simply make sure that you're logged in as a hiring manager and create a new position. As soon as you return to the detail page for the position, you'll see that the Unclaimed Positions Queue has automatically been assigned as the record owner, and that any user assigned to the Recruiter or Recruiting Manager role will have received an email notification.

To actually view the contents of the queue, click the Positions tab, and choose Unclaimed Positions Queue from the `View` drop-down list. Any recruiter or recruiting manager can click the **Accept** button on this page and take ownership of the new position.

Although our Assign Unclaimed Position Record to Recruiter workflow task won't be activated for another three days, we can see that it's currently scheduled to fire by checking out the *workflow queue*. This queue lists all of the time-dependent workflow actions that are scheduled for the future. To view it:

1. Log back in to your organization as an administrator.
2. Click **Setup ➤ Monitoring ➤ Monitor the Workflow Queue**.
3. Click **Search**.

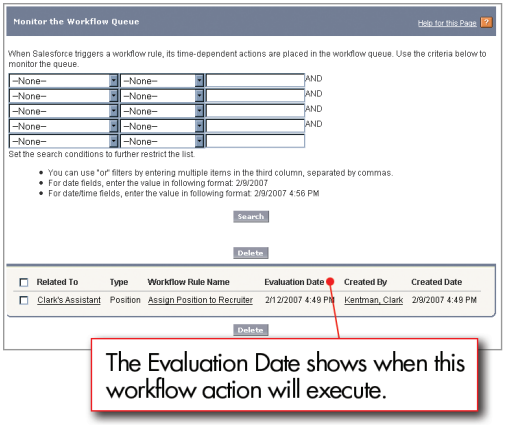


Figure 71: The Monitor the Workflow Queue Page

As soon as a recruiter takes ownership of the new Position record, this task is deleted from the workflow queue. Pretty slick, isn't it?

Now let's build one final workflow rule so we can see how to create a workflow email alert.

Creating a Workflow Rule That Sends Email Alerts

For our final workflow use case, let's create a workflow rule and email alert that sends a notification whenever a new Position is created. We want all the employees at Universal Containers to know when there are new positions available so they have the best opportunity to bring in referrals.

For this workflow rule, there's a step that we'll need to handle first: we need to design a template for what the email alert should look like.

Introducing Email Templates

Just as the platform includes built-in tools for setting security permissions, tracking events and tasks, and building business logic with workflow, it also provides a built-in tool for writing emails to users and contacts in your organization. *Email templates* allow you to create form emails that communicate a standard message, such as a welcome letter to new employees or an acknowledgement that a customer service request has been received.

To personalize the content of an email template, we can use *merge fields* to incorporate values from records that are stored in the system. For example, if we wanted to address an email recipient by their first name, we could write an email template as follows:

```
Dear {!Contact.FirstName},

...
```

In this example, `{!Contact.FirstName}` is a merge field that brings in the first name of the contact to whom the email is addressed, so an email to John Smiley would read:

```
Dear John,

...
```

For our workflow alert, we can build an email template to notify users of new Positions that have been added to the system. We can use merge fields to include information from the Position record, such as its title and the required skills. Let's go do that now, and then we can get back to finishing up our final workflow rule.

Try It Out: Building an Email Template

To build a new email template, we have to go to the Administration Setup area:

1. Click **Setup ➤ Communication Templates ➤ Email Templates**.

Here you should see a list of all the email templates that have already been defined for your organization, including several sample templates from salesforce.com.

2. Click **New Template**.

We can choose to create a text, HTML, or custom email template. HTML and custom email templates are the same except that HTML templates allow you to specify a letterhead to give your email the same look and feel as other emails from the same source. To keep things simple, we'll just stick with a plain text email for now.

3. Select `Text`, and click **Next**.

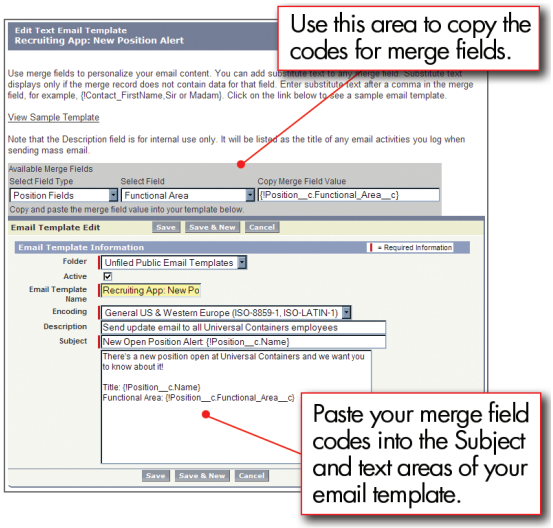


Figure 72: Defining an Email Template

The New Template page allows us to define the email template itself. The gray area near the top is where we'll generate the merge field codes for the fields in the email template below it, so let's skip past it for now and start with the `Folder` drop-down list.

4. In the `Folder` drop-down list, choose Unfiled Public Email Templates.

The Unfiled Public Email Templates folder is a standard, public folder available in every organization. By keeping the email template in a public folder, it'll be available to other users who have permission to view and edit email templates.

5. Select the `Available For Use` checkbox.

This option will make our email template available when we create our workflow alert.

6. In the `Email Template Name` text box, enter Recruiting App: New Position Alert.

Tip: To help keep your email templates organized, it's a good idea to preface any template name with the name of the app that uses it. Or, even better, you can create a public email template folder with the name of the app, such as Recruiting App Templates, and file all the relevant email templates in it.

7. In the `Encoding` text box, accept the default of General US & Western Europe (ISO-8859-1, ISO-Latin-1).

8. In the `Description` text box, enter "Send update email to all Universal Containers employees."

Now we get to the heart of our email template—the email's subject and body text.

9. In the `Subject` text box, enter New Open Position Alert: .

We want to put the title of the new position in the subject of our email, so we'll need to use our first merge field here, just after the colon in our subject. To get the proper code, we'll have to go back to the gray merge field area near the top of the page.

10. In the `Select Field Type` drop-down list, choose Position Fields.

Although there are many objects to choose from in the `Select Field Type` drop-down list, because we're creating an email template for a workflow rule, we're limited to the fields for the object that will be associated with that workflow—in our case, Position. That's because the workflow rule that uses this email template won't know about any individual records other than the Position record that triggered the rule's execution. If we put in fields from another object, they'd be left blank in our email, because there wouldn't be a record from which to pull the values.

Now let's grab the field we want.

11. In the `Select Field` drop-down list, choose Position Title.

In the `Copy Merge Field Value` text box, a merge field code appears for Position Title. We can cut and paste it to the end of our subject line so the subject now looks like this: New Open Position Alert: {!Position__c.Name}. When an email is generated from this template, `{!Position__c.Name}` will be replaced with the relevant position title.

Easy, right? Now let's finish the remainder of our email.

12. In the text area just below the `Subject` text box, enter the following text:

```
There's a new position open at Universal Containers!

Title: {!Position__c.Name}
Functional Area: {!Position__c.Functional_Area__c}
Location: {!Position__c.Location__c}

Job Description
{!Position__c.Job_Description__c}

Responsibilities
{!Position__c.Responsibilities__c}

Skills Required
```

```
{!Position__c.Skills_Required__c}

Educational Requirements
{!Position__c.Educational_Requirements__c}

If you know of anyone great who might be able to fill this role, please
 contact the hiring manager, {!Position__c.Hiring_Manager__c}.

Thanks!
```

13. Click **Save**.

That's it for our email template. Now that it's done, we're ready to create our New Position workflow rule and alert.

Try It Out: Creating the New Position Workflow Rule and Alert

Now that we've built our email template, we're ready to build the workflow rule and email alert that use it. By now this procedure should be very familiar to you:

1. Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Workflow Rules**.
2. Click **New Rule**.
3. In the `Select object` drop-down list, choose Position and click **Next**.
4. In the `Rule Name` text box, enter Email New Position Alert.
5. In the `Description` text box, enter "Send an email to everyone whenever a Position record is opened."

We only want this rule to execute once, whenever a Position record's status is set to Open - Approved.

6. In the Evaluation Criteria section, select `When a record is created, or when a record is edited and did not previously meet the rule criteria`.
7. In the first row of rule criteria:
   - Set the `Field` column to Status.
   - Set the `Operator` column to equals.
   - Set the `Value` column to Open - Approved. Although "New Position" is the default value of the `Status` picklist, we only want to publicize those positions that have been approved for hire by an executive.
8. Click **Save & Next**.

Now let's create the email alert for this workflow rule:

9. In the Immediate Workflow Actions area, click **Add Workflow Action**, and select **New Email**.
10. In the `Description` text box, enter Email New Position Alert.
11. Next to the `Email Template` field, click the lookup icon (), and select Recruiting App: New Position Alert.

We want to send this email to everyone at Universal Containers, but for this workflow rule there's no obvious way of doing that. We can work around this by relying on our role hierarchy and sending the email to everyone in the CEO role and its subordinates.

12. In the Recipient Type `Search` field, choose Role and Subordinates.
13. In the Available Recipients list, select Role and Subordinates: CEO and click **Add**.
14. Click **Save**.
15. Click **Done**.
16. Click **Activate**.

And that's all there is to it! To test out this workflow rule, all you need to do is create a new Position record with a `Status` value of Open - Approved. Within a few seconds, all users within your organization will receive an email letting them know that a position has just been created. Go ahead—try it!

Introducing Approvals

Now that we've just made a handful of workflow rules, let's take a look at another business logic tool that the platform provides: *approval processes*.

Approval processes allow you to specify a sequence of steps that are required to approve a new record. Each step allows a designated approver to accept or reject a record. The steps can apply to all records included in the process, or just to records that meet certain requirements. Like workflow, approval processes also allow you to specify actions—like sending an email alert, updating a field value, or assigning a task—that should occur whenever a record is approved, rejected, or first submitted for approval.

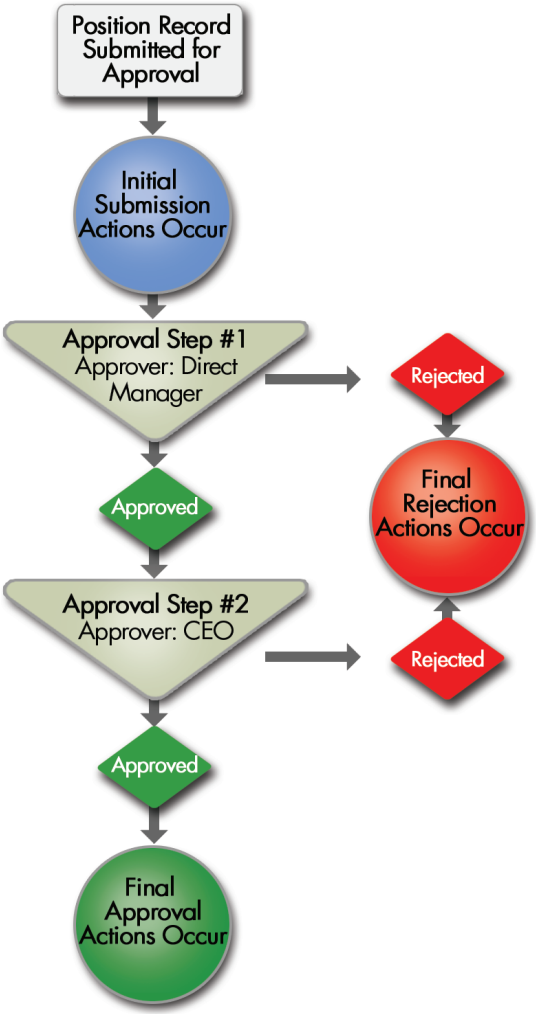


Figure 73: Approval Processes Consist of Steps and Actions

For example, suppose your organization has a three-tier process for approving expenses: submitted expenses that are less than $50 are automatically approved, those over $50 must be approved by a manager, and those over $5,000 must also be approved by a Vice President. For this example, you can define an approval process that specifies the following:

- If an expense record is submitted for approval, lock the record so that users cannot edit it and change the status to "Submitted."
- If the amount is $50 or less, automatically approve the request.
- If the amount is greater than $50, send an approval request to the direct manager.

- If the amount is greater than $5,000 and the first approval request is approved, send an approval request to the Vice President.
- If all approval requests are approved, change the status to "Approved" and unlock the record.
- If any approval requests are rejected, change the status to "Rejected" and unlock the record.

Neat, huh? For our recruiting app, we're going to define a similar approval process to submit new positions for approval. We want to make sure that a manager approves any position that his or her employee creates, and that any position with a minimum salary of more than $150,000 is approved by the CEO. Let's get started.

Planning for Approval Processes

Before we can actually define the approval process itself, there are a couple preliminary steps that we'll need to take care of first:

- Find a way to represent a user's direct manager. The direct manager will be the designated approver for the first step of our approval process.
- Define an email template that can be used to notify the designated approver that he or she has a pending approval request.

Although these preliminary steps are specific to our particular approval process, in most cases you'll find that you have to do a little advance planning before implementing an approval process on your own. The *Getting Started with Approval Processes* checklist available from the Tips and User Guides area of the online help outlines the things that you should think about and the components you should build before diving in.

Try It Out: Representing a User's Direct Manager with Hierarchical Relationship Fields

The first preliminary step for defining our approval process is to determine how we should represent a user's direct manager. This direct manager will be the designated approver for the first step of our approval process.

While at first it's easy to think that we can do this using a custom lookup field on the standard User object, if we look a little closer we'll discover some problems with this method. For one thing, a lookup field would allow a user's direct manager to be the same person as the user! While we might be able to work around this issue with a validation rule, the hairy issue of indirect associations comes up: how do we prevent a situation where user A is the manager of user B, who's the manager of user A? Or a situation where A is the manager of B is the manager of C is the manager of A? Clearly using a lookup for the `Direct Manager` custom field leaves room for data inconsistencies that we won't be able to handle with simple validation rules.

Fortunately, the platform comes to the rescue once again with a *hierarchical relationship* custom field type specifically designed for the User object. This field type was created specifically for a situation like ours in which we want to associate one user with another without indirectly associating that user to him or herself. With the hierarchical relationship field type, the platform takes care of all of the error checking for us.

Now that we know about hierarchical relationships, creating the `Direct Manager` field is just as easy as creating any other custom field:

1. Click **Setup ➤ Customize ➤ Users ➤ Fields**.
2. In the User Custom Fields related list, click **New**.
3. Select `Hierarchical Relationship` and click **Next**.
4. In the `Field Label` field, enter Direct Manager.
5. Select `Restricted Field`.

We'll select the `Restricted Field` checkbox because users shouldn't be able to change the name of their own manager.

6. Click **Next**.
7. Click **Save**.

Great! Now on to our email template.

Try It Out: Creating an Email Template for Approvals

Our second preliminary task involves creating an email template to notify designated approvers that a record needs to be approved. Since we've already talked about email templates in *Introducing Email Templates* on page 172 we'll simply include the values that we want to use for the template in the following table. You can build it from **Setup ➤ Communication Templates ➤ Email Templates**.

Table 36: The "Recruiting App: New Position Requires Approval" Email Template

Parameter	Value
Template Type	Text
Available For Use	Selected
Email Template Name	Recruiting App: New Position Requires Approval
Encoding	General US & Western Europe (ISO-8859-1, ISO-LATIN-1)

Parameter	Value
Description	Send notification email to designated approver when new position record requires approval.
Subject	New Position Requires Approval

Email Body	`A new position record has been submitted for your approval. Please visit the link below and either approve or reject it.`
	
	`{!Position__c.Link}`
	
	`Thanks!`

Try It Out: Creating an Approval Process

Now that we've finished our preliminary tasks, we're ready to define the approval process itself. The approval process definition acts as a framework for the approval steps and actions that we'll define later:

1. Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Approval Processes**.
2. From the **Manage Approval Processes For** drop-down list, choose Position.

There are two different wizards that we can use to create a new approval process: a Jump Start Wizard and the Standard Setup Wizard. The Jump Start Wizard sets several default values for us and only requires input for the most crucial fields: the approval assignment email template, filter criteria to enter the approval process, and the designated approver. The Standard Setup Wizard, on the other hand, allows us to configure every possible option for our approval process. We'll stick with it for now so we can take a look at all of the options that are available.

3. From the **Create New Approval Process** drop-down button, choose **Use Standard Setup Wizard**.
4. In the `Process Name` field, enter Approve New Position.
5. In the `Description` field, enter "Ensure that a manager approves any position that his or her employee creates, and that any position with a minimum salary of more than $150,000 is approved by the CEO."
6. Click **Next**.

After specifying the name and description, our next step is to enter the filter criteria that should be used to determine which positions need approval. In our case, we want all positions created by a user other than the CEO to get approved by at least a direct manager.

7. In the first row of filter criteria, select Current User: Role not equal to CEO.
8. Click **Next**.
9. From the `Next Automated Approver Determined By` drop-down list, select Direct Manager.

Note: Rather than creating our `Direct Manager` field ahead of time, we could have selected Create New Field from this drop-down list to define a new custom hierarchical lookup field on the fly.

Figure 74: Specifying the Approver Field

10. In the Record Editability Properties area, choose `Administrators ONLY can edit records during the approval process`.

Record editability allows you to specify whether a record that's been submitted for approval can be edited by the approver before being approved. Since we don't want managers to change the positions that a hiring manager or recruiter creates without alerting the owner, we'll only let administrators perform edits while a record is in our approval process.

11. Click **Next**.
12. In the `Approval Assignment Email Template` lookup field, select Recruiting App: New Position Requires Approval.
13. Click **Next**.

Our next step in defining the approval process is specifying which fields should be displayed when an approver approves or rejects a record. We can display any fields on the Position object, but since we've restricted the editability, these fields can't be edited by the approver until the record is approved or rejected.

14. Move the following fields from `Available Fields` to `Selected Fields`:
    - Position Title
    - Owner

- Hiring Manager
- Type
- Location
- Hire By
- Job Description
- Min Pay
- Max Pay

Figure 75: Defining the Record Approval Page Layout

On this page we can also specify whether approval history information should be displayed. This information shows whether this record was submitted for approval in the past, who was the designated approver, and whether it was approved or rejected.

15. Select `Display approval history information in addition to the fields selected above`.

Finally, before leaving this page, we can specify security settings to determine whether an approver can approve or reject records from a wireless-enabled mobile device. Unless it's a mandatory requirement for your approvers, it's better not to choose this option because it prevents a user from manually selecting an appropriate approver for a record. We'll leave the default choice selected for now.

16. Click **Next**.

The last page of the New Approval Process wizard allows us to choose who should be allowed to submit Position records for approval. Again, we'll just leave the default Record Owner selected, because there's no reason for another user to have this power.

The last two options on this page allow us to place the Approval History related list on all Position page layouts and give users the ability to recall pending approval requests after they've submitted them. The Approval History related list is the same history related list that we included on the approver page layout, so we'll also include it on the Position detail page. From this related list users can also click the **Recall Approval Request** button to withdraw their pending approval requests. If we didn't enable this last option, only administrators would have access to the **Recall Approval Request** button.

17. Select `Add Approval History Related List to All Position Page Layouts`.
18. Select `Allow submitters to recall approval requests`.
19. Click **Save**.

Phew! We've finished defining the framework for our approval process, but we won't be able to activate it until we've given it some steps and some actions to fire when records are actually approved or rejected. Let's move on to those now.

20. Select `Yes, I'd like to create an approval step now`.
21. Click **Go!**

Try It Out: Creating Approval Steps

As we said earlier, every approval process consists of a set of steps that are required to approve the creation of a new record, and each step allows a designated approver to accept or reject a record. In other words, every "signature" that you need to get a record approved must have a corresponding step in the approval process. An approval step consists of:

- A designated approver
- Optional filter criteria, so that only records that meet certain conditions will require approval at that step
- Optional step approval actions that execute regardless of the outcome of the whole approval process
- Optional step rejection actions that execute regardless of the outcome of the whole approval process

For our New Position approval process, we'll need to define two steps—one that requires approval from a direct manager for all new Position records, and one that requires additional

approval from the CEO for Position records with minimum salaries that fall above $150,000. Let's define the first step for all new Position records now.

Note: Because we selected `Yes, I'd like to create an approval step now` at the end of the Standard Approval Process wizard in the last section, we're already at the beginning of the New Approval Step wizard. If we weren't, we could return to the same wizard with the following steps:

- Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Approval Processes**.
- In the Inactive Approval Processes related list, click **Approve New Position**.
- In the Approval Steps related list, click **New Approval Step**.

1. In the `Name` field, enter Direct Manager Approval.
2. In the `Description` field, enter "Every new Position record must be approved by the Position owner's direct manager."
3. In the `Step Number` field, enter 1.

The `Step Number` field specifies the order in which multiple steps should be processed. By assigning this as Step 1, it will be the first to execute when the approval process is triggered.

4. Click **Next**.

The Specify Step Criteria area allows us to filter the records that require approval during this step. Because we've already filtered out Position records that are owned by the CEO from the whole approval process, this step does not need any additional filtering.

5. Click **Next**.

Finally we have to select the assigned approver for this step, and specify whether his or her delegate is allowed to approve the request as well. Because this is the Direct Manager approval step, it clearly makes sense to accept the default option of `Automatically assign using the custom field selected earlier. (Direct Manager)`. However, because Position records aren't particularly sensitive, it's okay for managers to assign delegate approvers. So managers who go on vacation, or who receive large quantities of approval requests, can share their work with another employee.

6. Select `The approver's delegate may also approve this request`.
7. Click **Save**.

Having completed our first approval step, we're faced with another choice to create optional approval or rejection actions for this step, or to return to the approval process detail page. While we ultimately need to specify *final* approval and rejection actions that occur when the approval process ends one way or the other, there's nothing in particular that needs to happen

after this first step that we can't specify elsewhere. Let's return to the detail page for our approval process and define our second approval step for positions with minimum salaries of more than $150,000.

8. Select `No, I'll do this later. Take me to the approval process detail page to review what I've just created.`
9. Click **Go!**.
10. In the Approval Steps related list, click **New Approval Step**.

Once again we're back in the New Approval Step wizard, but this time it includes a summary of the previous step that we created. This helps us to remember where we are in our approval process.

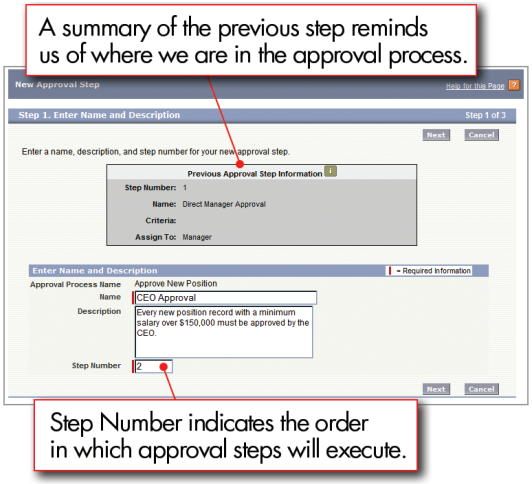


Figure 76: Defining a Second Approval Step

11. In the `Name` field, enter CEO Approval.
12. In the `Description` field, enter "Every new Position record with a minimum salary over $150,000 must be approved by the CEO."
13. In the `Step Number` field, enter 2.
14. Click **Next**.

For this approval step, we only want to send positions with a minimum salary over $150,000 to the CEO. Additionally, we want to exclude any records that the CEO has already approved (for example, because one of the CEO's direct reports created the record).

15. Select the `Enter this step if the following` radio button, then choose `criteria are met` from the drop-down list.
16. In the first row of filters, enter Min Pay greater or equal 150000.

Tip: As a short cut, you can specify 150,000 as 150k.

17. In the second row of filters, enter Current User: Direct Manager not equal to Cynthia Capobianco (the acting CEO in *Figure 58: Universal Containers Role Hierarchy* on page 137).
18. Click **Next**.

Finally, we need to select the approver (the CEO), and specify what should happen if he or she rejects this request.

19. Select the `Assign to` radio button, select User, click the lookup icon (), and choose the name of the CEO in your organization (Cynthia Capobianco).
20. Select `The approver's delegate may also approve this request`.

The next section allows us to specify what to do with the record if it's rejected at this step. Because the Position record is locked from editing during the approval process, it makes the most sense to perform the final rejection.

21. Select `Perform all rejection actions for this step AND all final rejection actions. (Final Rejection)`.
22. Click **Save**.

Once again, we're faced with a choice to define approval or rejection actions for this particular step. Let's circumvent those, and return to the approval process detail page to define our initial submission, final approval, and final rejection actions for the whole process.

23. Select `No, I'll do this later. Take me to the approval process detail page to review what I've just created`.
24. Click **Go!**.

Try It Out: Creating Approval Actions

Now that we've finished defining our approval process steps, we're nearly done. All that remains is to specify the *approval actions* that should occur whenever a record is initially submitted, or when it's finally approved or rejected.

Just like workflow actions, approval actions allow you to create and assign tasks, update fields, and send email updates and outbound messages. They can either be associated with the approval process as a whole or with individual approval steps.

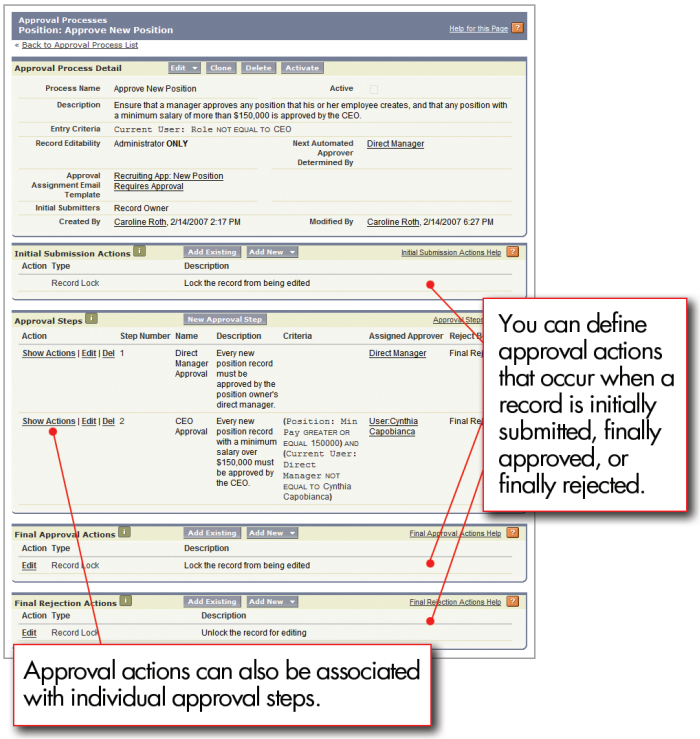


Figure 77: The Approval Process Detail Page

Because defining an approval action is almost identical to the way we created workflow actions, we'll quickly step through the process of updating the `Status` field to Pending Approval when a position is initially submitted and then leave our other approval actions as exercises:

1. If you're not on the approval process detail page already, click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Approval Processes**, and then click **Approve New Position**.
2. In the Initial Submission Actions related list, click **Add New**, and select **Field Update**.

Does the New Field Update Action page look familiar? That's because we've seen it before—all approval tasks, field updates, email alerts, and outbound messages use the same editing interface as workflow actions. In fact, every workflow and approval action that you create can be used interchangeably on both workflow and approval processes.

3. In the `Name` field, enter Set Status to Pending Approval.
4. In the `Description` field, enter "While a position is in an approval process, its status should be set to Pending Approval."
5. In the `Field to Update` drop-down list, select Status.

6. In the Picklist Options area, select `A specific value` and choose Pending Approval from the list.
7. Click **Save**.

Easy, right? Now, to finish up the rest of the approval process, define the remaining approval actions on your own according to the values in the following table.

Table 37: Additional Approval Actions

Category	Type	Values
Final Approval Actions	Field Update	Name: Set Status to Open Approved Field to Update: Status A specific value: Open - Approved
Final Rejection Actions	Field Update	Name: Set Status to Closed - Not Approved Field to Update: Status A specific value: Closed - Not Approved
Final Rejection Actions	Field Update	Name: Set Close Date to Today Field to Update: Close Date Use a formula to set the new value: TODAY()

Try It Out: Activating Our Approval Process

We're all done with defining our Approve New Position approval process, but in order to test it one final step remains: we've got to activate it. Before we do so, however, be careful! Once an approval process has been activated at least once, its approval steps can no longer be edited. The only way to make changes is to clone the existing approval process and make edits in the cloned one.

Once you're ready to test the approval process:

1. Click **Setup ➤ Customize ➤ Workflow & Approvals ➤ Approval Processes** to return to the approval process list page.
2. Click **Activate** next to the Approve New Position approval process.

The Approve New Position approval process automatically moves to the Active list, and a new field is displayed: `Process Order`. This field is important if we're trying to use more than one approval process at a time because it tells us the order in which each approval process will be evaluated.

Look At What We've Done

That approval process took a bit of work to set up, but let's take a look at what we've accomplished:

1. First click **Setup ➤ Manage Users ➤ Users** and edit the user record for your Recruiting Manager to fill in the `Direct Manager` field so the approval chain is properly set up.
2. Then log in to your app as a Recruiting Manager and create a new position.

Notice that after clicking **Save**, the detail page displays a **Submit for Approval** button in the new Approval History related list.

Figure 78: The Submit for Approval Button

3. Click **Submit for Approval** and then click **OK**.

Clicking the **Submit for Approval** button causes several things to happen. First, the record is locked from editing, as shown by the lock icon at the top of the page. Additionally, two new entries appear in the Approval History related list showing who submitted the record and the current approval assignment. Finally, the direct manager of the position owner receives an email reporting that there's a new position to approve.

Figure 79: A Submitted Position Record

When the direct manager next logs in and visits the record, an **Approve/Reject** button is visible on the Approval History related list. He or she can click this button to see the approval request and approve or reject the record, with comments.

4. Log in as the direct manager who is responsible for approving the request.
5. Click **Approve/Reject** in the Approval History related list on the record, enter any optional comments, and then click **Approve**.

Tip: To make accepting and rejecting approval requests more convenient, consider adding the Items to Approve related list to the default Home page layout:

1. Click **Setup ➤ Customize ➤ Home ➤ Home Page Layouts**.
2. Click **Edit** next to the Dashboard Home Page Default.
3. Select `Items to Approve` and click **Next**.
4. Arrange the Items to Approve component on the page layout by moving it up and down the `Wide (Right) Column` list as desired.
5. Click **Save**.

Figure 80: The Approval Request Page

If the approver accepts the record, it progresses to the next step of the approval process (if its `Min Pay` field is greater than $150,000 and the CEO still hasn't approved it), or else the position's `Status` field is set to Open - Approved. The record details remain locked to protect them from being changed, but recruiters can still associate the position with job applications, tasks, or other activities. If the record is rejected, the `Status` is set to Closed - Not Approved, the `Close Date` field is set to today's date, and the record is unlocked in case it just needs a simple edit before it reenters the approval process. With just a few minutes of work, we've built an effective business process that will make all of Universal Containers' users more effective.

Summing Up

Check out our Recruiting app now! By leveraging the platform's built-in workflow and approval process tools, we've transformed our app from a glorified database into a fully functional application that provides real value to its users.

Next we'll tackle something that provides real value to our executive users: reports and dashboards that give our users a complete analytical picture of how the recruiting program at Universal Containers is going.

Chapter 9

Analyzing Data with Reports and Dashboards

In this chapter ...

We've come a long way with our Recruiting app—not only do we have four custom objects to store our data, but we've also defined security and sharing rules to protect it, and we've implemented several business processes with workflow and approvals. We've built a functional on-demand application, and we haven't even written a single line of code!

Now it's time to turn our attention to the needs of the Universal Containers managers and executive staff. Because they need to keep track of on many different aspects of the business, we need a way to give them a bird's-eye view of the company's recruiting activity without forcing them to delve into piles and piles of data. To do this, we'll create a set of custom reports for our Recruiting app and then build a dashboard that allows users to view summaries of key Recruiting app statistics every time they log in.

Introducing Reports

We can help users monitor and analyze the data that's being generated in their organization by building *reports*. Reports are summaries of the data that's stored in an app. They consist primarily of a table of data, but can also include data filters, groupings, and a customized graph.

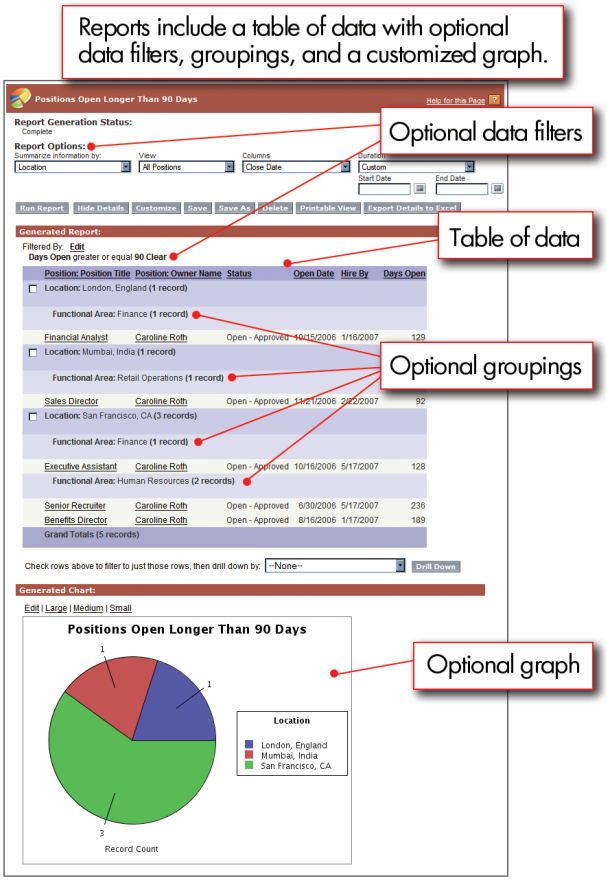


Figure 81: A Sample Report

While a comprehensive set of reports is included with every organization to provide information about standard objects such as contacts and accounts, we can also build custom reports that highlight interesting metrics about the data stored in our custom objects.

For example, some of the questions that an executive at Universal Containers might have about recruiting include:

- On average, how many days does it take for each recruiter to fill a position?

- Which functional areas have the most open positions?
- Which positions have been open for more than 90 days?
- Which positions are getting the most candidates?
- Which employees conduct the most interviews?
- What does the job application pipeline look like for each open position?
- Who have we hired in the last 90 days?

We can answer all of these questions and more by creating custom reports in the Reports tab of the app. Although this tab isn't visible by default in our Recruiting app, any user can click the arrow tab on the right side of the tab bar to display all available tabs, and then navigate to the reports area by clicking **Reports**.

Tip: If a user goes to the Reports tab on a regular basis, he or she can also add this tab to their main tab bar by clicking the arrow tab on the right side of the tab bar, selecting **Customize My Tabs**, and then moving Reports to the `Selected Tabs` list. This will save them a click since they'll no longer need to click the arrow tab to see the list of all tabs.

Report Types

The platform supports three different types of reports, each with varying degrees of functionality and complexity:

- *Tabular reports* are the simplest and fastest way to look at your data. Similar to a spreadsheet, they consist simply of an ordered set of fields in columns, with each matching record listed in a row. While easy to set up, they can't be used to create groups of data or graphs. Consequently, they're best used just for tasks such as generating a mailing list.

  Tip: Use tabular reports when you want a simple list or a list of items with a grand total.

- *Summary reports* are similar to tabular reports, except that they also allow you to group rows of data, view subtotals, and create graphs. For example, in the sample Employee Interviewer reports that appear in the following screenshot, the summary report groups the rows of reviews by the possible values of the `Owner Name` field, allowing us to see at a glance subtotals of how many times the two interviewers have talked to candidates and entered reviews for them.

  While a little more time-consuming to set up, summary reports give us many more options for manipulating and organizing the data, and, unlike tabular reports, they can be used in

dashboards. Summary reports are the workhorses of reporting—you'll find that most of your reports tend to be of this type.

Tip: Use summary reports when you want subtotals based on the value of a particular field or when you want to create a hierarchical list, such as sales organized by year and then by quarter.

- *Matrix reports* are the most complex kind of report available, allowing you to group records both by row and by column. For example, in the following sample Employee Interviewer reports, the matrix report groups the review rows by the possible values of the `Owner Name` field, and also breaks out the possible values of the `Position` field into columns. Consequently, the report gives us subtotals for the number of times an interviewer has interviewed candidates and entered reviews for a particular position. These reports are the most time-consuming to set up, but they also provide the most detailed view of our data. Like summary reports, matrix reports can be used in dashboards.

Tip: Use matrix reports when you want to see data by two different dimensions that aren't related, such as date and product.

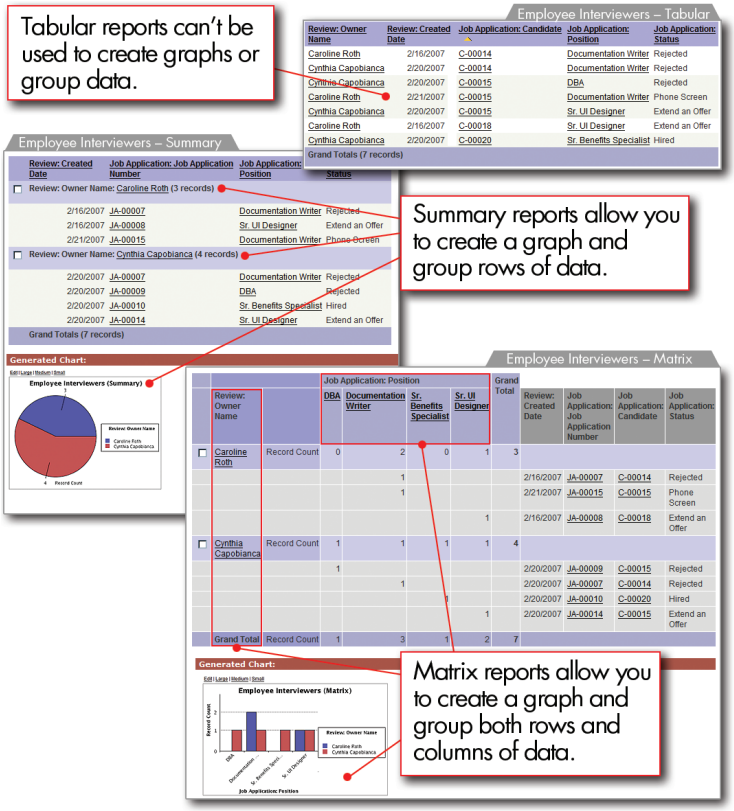


Figure 82: Tabular, Summary, and Matrix Reports Offer Different Options for Viewing the Same Data

Setting Up the Recruiting App for Reports

Before we get started building reports, we first need to take care of a couple tasks that will make our reports easy to find in the Recruiting app. We need to add the Reports tab to the default tab display along with our Positions, Candidates, and Job Applications tabs, and we'll also need to create a folder for storing all the reports that we create. While both of these tasks are purely optional, they'll go a long way towards improving the experience of our Recruiting app users.

Try It Out: Adding the Reports Tab

First we'll start by adding the Reports tab to the set of default tabs that are displayed for every Recruiting app user. To do so, we need to revisit the Recruiting app that we created way back in *Chapter 4: Building a Simple App* on page 31:

1. Click **Setup ➤ Build ➤ Custom Apps**.
2. Click **Edit** next to the Recruiting app.
3. In the Choose the Tabs section, add Reports to the `Selected Tabs` list.

You can select the `Overwrite users' personal custom app customizations` option so that the Reports tab is automatically added to the tab bar by default for all users. Just remember that this option will overwrite any changes that current users may have already made. If you've already deployed your app and you'd rather not affect existing users, you can leave this option unchecked, but users will have to manually add the Reports tab to their personal tab bar by clicking the arrow tab on the right side of the tab bar, selecting **Customize My Tabs**, and then moving Reports to the `Selected Tabs` list.

4. Optionally, select `Overwrite users' personal custom app customizations`.
5. Click **Save**.

Perfect! Now let's visit the Reports tab to perform our next task: creating a folder for Recruiting reports.

Try It Out: Creating a Recruiting Reports Folder

When you go to the Reports tab, you'll see that it displays a long list of all the standard and custom reports that are defined for your organization. As you can see by their headings, they're divided into categories to help users find what they're looking for. You can also quickly jump to a particular category by choosing it from the `Folder` drop-down list.

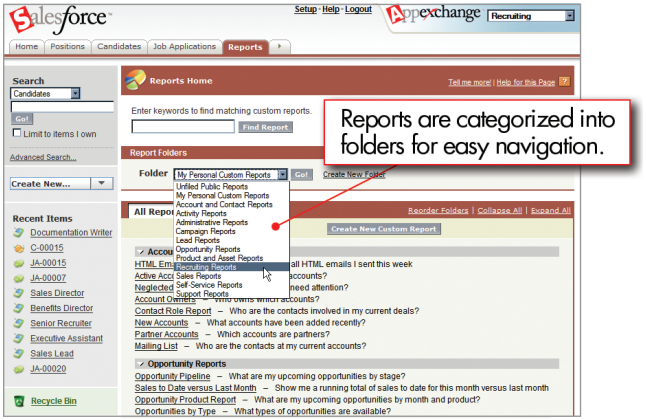


Figure 83: The Reports Tab and the Folder Drop-Down List

Because every organization already comes with over fifty standard reports and around a dozen report categories, it's important to create a new category whenever you're defining a new set of reports. It's an easy way of making sure that your users can always find the reports they need. To define a new report category, we simply have to define a new report folder:

1. In the Reports tab, click **Create New Folder** next to the `Folder` drop-down list.
2. In the `Report Folder` field, enter Recruiting Reports.
3. In the `Public Folder Access` drop-down list, choose Read Only.

By choosing Read Only for this folder, only administrator users can modify the reports the folder contains, or save new reports to it. However, because all users can create their own reports, this won't hinder your users from modifying a report that we create and saving it to a personal folder of their own.

4. Select `This folder is accessible by all users`, which allows all users to view this folder and the reports in it.
5. Click **Save**.

Great—we're now ready to create our first report.

Creating a Summary Report

To get started with creating custom reports, we'll first create a summary report that answers the question, "Which functional areas have the most new or open positions?" This report will include the title, hiring manager, location, and status of each position, as well as a pie chart that shows a visual representation of the data. As we do this, we'll skip over some of the more complicated reporting options so we can quickly create a report that fits our requirements. Our

second example will then define a more complex matrix report that shows off some of the more advanced reporting features.

Because this report is going to use so many different reporting features, we'll break down the procedure into three parts:

1. Creating a summary report
2. Adding columns and filters
3. Adding a chart

Try It Out: Creating a Summary Report

To create our summary report, we'll start by opening the custom report wizard:

1. On the Reports tab, click the **Create New Custom Report** button.

Before we can enter the heart of the actual wizard, we first have to choose the object or relationship that should be the focus of the report. The primary object that we choose in this screen specifies the records that will be counted, and the available fields that we can use to filter, group, or display in the report.

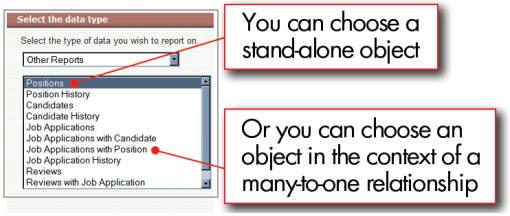


Figure 84: Selecting the Report Data Type

To help with navigation in this screen, all objects and relationships are grouped in categories like Accounts & Contacts or Customer Support Reports. The custom objects and relationships that we built for our Recruiting app are placed under the Other Reports category.

Note: The Other Reports category contains all reports that are based only on custom objects. If you've built a custom object that's related to a standard object, such as Account or Contact, you'll also be able to report on your custom object in the standard object's category.

2. From the drop-down list, choose Other Reports.

Objects that have a many-to-one relationship with another object, like Job Applications and Positions, can either by selected on their own or in the context of their relationship with the

other object. For example, if we select Job Applications with Position, our report will count Job Application records, but can filter, group, or display fields from the related Position records as well. This will come in handy a little later when we build reports that count Job Application records. But because we need to count Position records in our report and Positions aren't on the many side of a relationship, we'll stick with a stand-alone Positions report for now.

3. Select Positions.
4. Click **Next**.

Now that we've chosen the Position object, we're finally able to enter the report wizard. From this point onwards, we can jump back and forth to various wizard steps, but we can no longer return to the original object selection screen unless we cancel out of the wizard and start over again. While in the wizard, we can also run the report at any time to see if we have the results we're looking for. For example, let's see what a baseline Positions report looks like without any customizations.

5. Click **Run Report**.

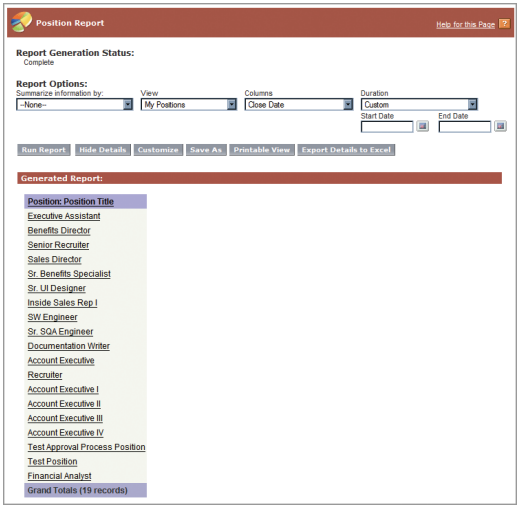


Figure 85: A Positions Report with No Customizations

Not too bad, huh? Without specifying any details, we've already got a list of Position records with a Grand Total at the bottom. This is the equivalent of what we might see if we were creating a tabular report without additional columns. Now let's take this basic report to the next level.

6. Click the **Back** button in your Web browser to return to the report wizard.

In our first wizard step, we need to choose the type of report that we want to create. Because we want to group rows of open positions by functional area but don't need to define additional groupings in the columns dimension, we'll create a summary report.

7. Select `Summary Report`.
8. Click **Next**.

The next step of the wizard allows us to specify which numerical or checkbox field values should be included in our report, and how each of them should be summarized in our report subtotals and grand totals. We'll be talking more about what you can do on this page when we create our second report. For now, because we're not including any numerical field values other than record counts (which are already included in summary reports by default), we can skip to the next wizard step.

9. Click **Next**.

The third step of the wizard allows us to choose up to three different fields by which to group our rows of data. In this case, we want to group by the `Functional Area` field.

10. In the `Summarize Information by` drop-down list, choose Functional Area.
11. Click **Next**.

Once again, let's run our report to see what we've done.

12. Click **Run Report**.

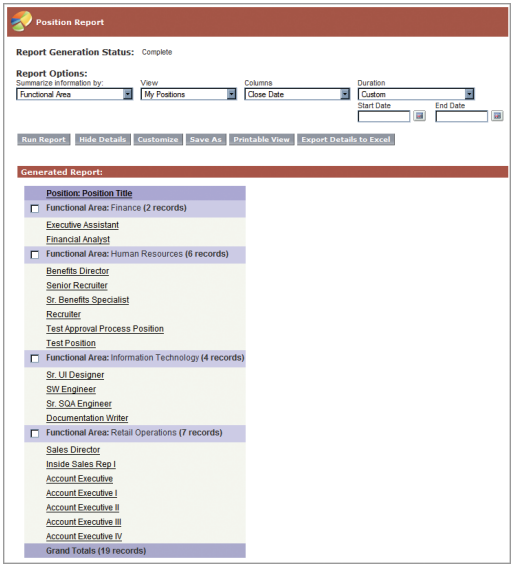


Figure 86: A Positions Report with Grouping

We're closer now! Our report still only lists position titles, but now they're grouped by the functional area to which they belong. Also, each grouping has a record count subtotal in parentheses. These counts will be the basis of the pie chart that we'll add later.

13. Click the **Back** button in your Web browser to return to the report wizard.

Tip: When you're viewing a report, you can always click the **Customize** button to get back into the report wizard. You may not always go back to the step you want, but you can use the `Jump to Step` drop-down list at the top of the report wizard to quickly go to the step of the wizard you want.

Try It Out: Adding Columns and Filters

The next three steps of the wizard allow us to select and order the fields that are displayed in our report, and set the filters by which we can restrict the set of records included in the report.

In addition to `Position Title`, which is already selected by default, we also want to display the `Hiring Manager`, `Location`, and `Status` fields for each record.

1. In the Position Info area, select:

- `Position: Position Title`
- `Hiring Manager`

- `Location`
- `Status`

2. Click **Next**.
3. In the Report Column Order area, reorganize the order of the columns as you see fit. This is the order of the columns as they'll display in the report from left to right, so the field listed in the top position on this page of the wizard will be the left-most column in the report.
4. Click **Next**.

Let's run our report one more time.

5. Click **Run Report**.

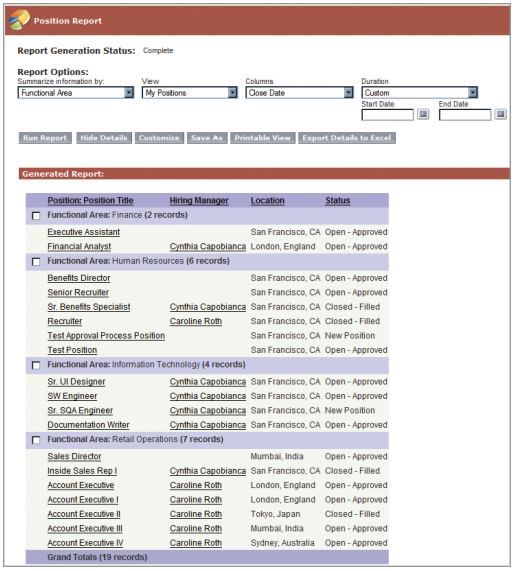


Figure 87: A Positions Report with Grouping and Additional Columns

We're even closer to our goal. However, the `Status` field shows us that the report includes some Position records that have already been filled. Since we only want to view the new and open ones, we'll need to set some filters. Conveniently, we can set these in the next step of the wizard.

6. Click the **Back** button in your Web browser to return to the report wizard.

The sixth step of the wizard allows us to set the filters that should be applied to this report. Filters define the set of records to be included in the report; for example, you include only records created this month or only records belong to a certain user. The Standard Filters area

lets us quickly filter by record owner and any date field, while the Advanced Filters area allows us to filter on any field value. Because we want to view open positions across the entire organization and not just positions that we own (by default, all custom reports include "My" records only), we need to set two filters:

7. In the `View` drop-down list, choose All Positions.
8. In the first row of the Advanced Filters area, define a filter for "Status equals New Position, Pending Approval, Open - Approved."

Tip: Notice that whenever you choose a checkbox field or a picklist field, like `Status`, in your filter, a lookup icon () becomes available next to the filter row. You can click the lookup icon to view valid values for that field and quickly insert the ones by which you want to filter.

Using this filter of "Status equals New Position, Pending Approval, Open - Approved" means that our report will include only those Position records with one of these three statuses. Note that the comma between the three `Status` values is treated as an OR, so this one filter is the same as using these three filters:

- Status equals New Position, OR
- Status equals Pending Approval, OR
- Status equals Open - Approved

9. Click **Next**.

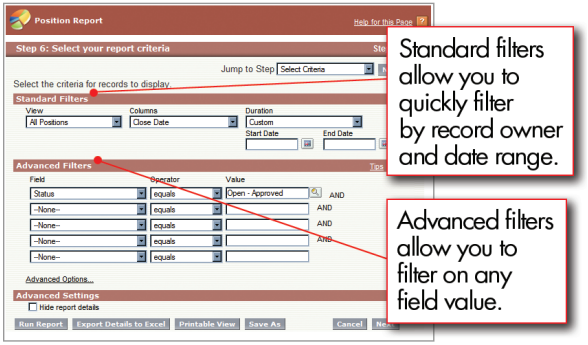


Figure 88: Setting Report Filters

Try It Out: Adding a Pie Chart

The final step of the wizard allows us to specify the chart that should be displayed under the report data. In our case, we want a pie chart:

1. In the `Chart Type` drop-down list, select Pie.

The wizard automatically knows that we want the values to be the record counts, and the wedges to be the different functional areas. That's because chart values always correlate to the numerical metrics that appear in the report, and pie chart wedges always correlate to a report's groupings. Had we grouped our data with an additional column (like `Position Owner`), we would have had a choice of which field values to display in the wedges.

Now let's finish up our chart and generate our final report:

2. In the `Chart Title` field, enter Open Positions by Functional Area.
3. Click **Run Report**.

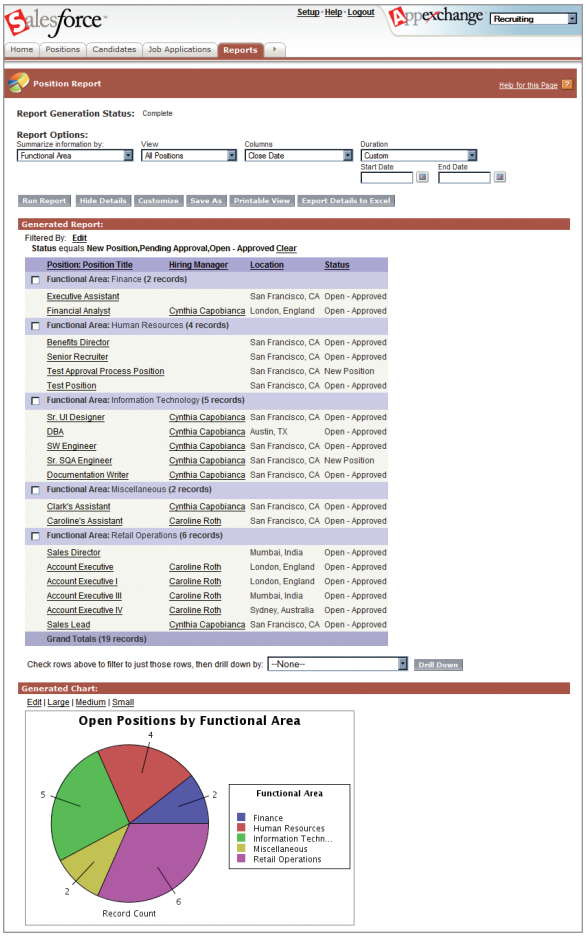


Figure 89: The Final "Open Positions by Functional Area" Summary Report

Terrific! Because our report meets all of our criteria, let's save it to the Recruiting Reports folder.

4. Near the top of the report, click **Save As**.
5. In the `Report Name` field, enter Open Positions by Functional Area.
6. In the `Report Description` field, enter "Which functional areas have the most new or open positions?"
7. From the `Report Folder` drop-down list, select Recruiting Reports.
8. Click **Save**.

Now if we view the Recruiting Reports folder, we can see our new summary report. Next, we'll make a matrix report that takes advantage of some of the more advanced reporting features.

Creating a Matrix Report with Custom Field Summaries, Time-Based Filters, and Conditional Highlighting

For our next report, we'll answer the question, "On average, how many days does it take each recruiter to fill a position with or without required travel?" This report will use a matrix format to highlight the difference that mandatory travel makes in how long it takes for positions to be filled. Looking at data only for this year and the last, the report will include:

- Record counts for each position a recruiter owns
- A custom field summary for the percentage of a recruiter's positions that require travel
- A time-based filter that restricts the data only to positions that were created this year or last
- A color-coded field summary for the average number of days positions remain open:
    - Averages of less than 30 days are color-coded green
    - Averages of between 30 and 60 days are color-coded yellow
    - Averages of more than 60 days are color-coded red

Because this report is going to use so many different reporting features, we'll break down the procedure into four parts:

1. Creating a matrix report
2. Adding custom summary fields
3. Adding columns and filters
4. Adding a chart and conditional highlighting

Try It Out: Creating a Matrix Report

To create the matrix report, we'll step through the custom report wizard again, this time highlighting steps that are different from how we defined the summary report.

1. In the Reports tab, click **Create New Custom Report**.

Once again, we'll be creating a report that counts Position records.

2. From the drop-down list, choose Other Reports.
3. Select Positions, and click **Next**.

Because we want to directly compare individual recruiter performance for positions that do and do not require travel, we'll use a matrix report. That way we can group the rows of positions by recruiter, and columns of positions by whether or not they require travel.

4. Select `Matrix Report`, and click **Next**.

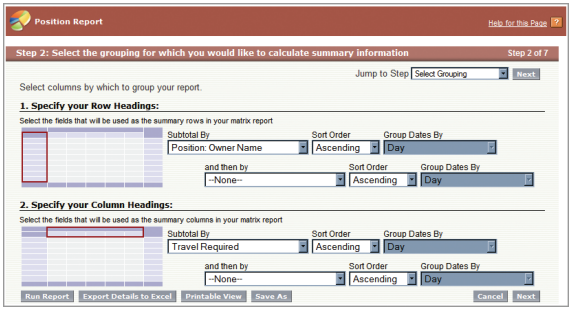


Figure 90: Specifying Groupings in a Matrix Report

The Grouping step of the report wizard now allows us to group by both rows and columns.

5. Under Specify your Row Headings, select Position: Owner Name from the `Subtotal By` drop-down list.
6. Under Specify your Column Headings, select Travel Required from the `Subtotal By` drop-down list.
7. Click **Run Report**.

Our report now breaks out the possible values for the `Travel Required` field in the columns dimension. Recruiters are also broken out in the row dimension, but because all custom reports query just the report creator's data by default (that is, "My" records only), only one recruiter is listed in the report so far. Let's keep going.

8. Click the **Back** button in your Web browser to return to the report wizard.

9. Click **Next**.

Try It Out: Adding Custom Summary Fields

Step three of the report wizard allows us to specify which numerical or checkbox field values should be included in our report, and how each of them should be summarized in subtotals and grand totals. While the number of records that match report criteria are always summarized as a sum total, other numerical and checkbox columns are sometimes best summarized in a different way.

For example, it doesn't make much sense to sum up the values of the `Days Open` column—the resulting value would keep incrementing with every additional record that was returned and would be a meaningless piece of data. However, if all `Days Open` values in a report were averaged, the summary total would provide interesting information—the average number of days that each position has been open.

For our report, we need to include three different types of summary information: record count, average days open, and a custom summary field that specifies the percentage of records that require travel. While the first two are standard summary fields that the platform supports by default, the third will require a visit to the Custom Summary Formula editor. Let's start with the first two:

1. In the Record Count row, select the `Sum` checkbox.
2. In the Days Open row, select the `Average` checkbox.

Note: Because we don't need to include `Max Pay` or `Min Pay` in our report, we don't need to specify how they should be summarized. Likewise, because we're defining a custom summary formula for `Travel Required`, we also don't need to specify a standard summary field for it.

3. In the Custom Summary Formulas area, click **New**.

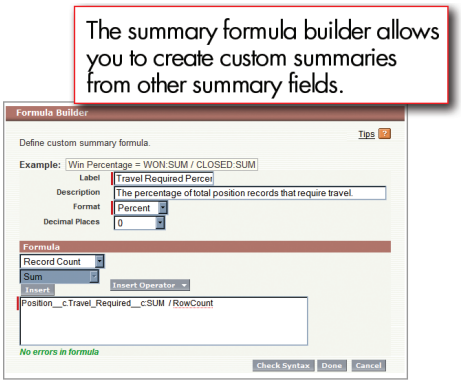


Figure 91: The Custom Summary Formula Builder

The Formula Builder allows us to define a new summary formula based on the other standard summary values that are available. In our case, we want to include a summary that shows the percentage of Position records that require travel in any given segment. To make this calculation we need to divide two sums—the sum of all records that require travel divided by the sum of all total records:

4. In the `Label` field, enter Travel Required Percentage.
5. In the `Description` field, enter "The percentage of total Position records that require travel."
6. In the `Format` drop-down list, choose Percent.
7. In the `Decimal` drop-down list, choose 0.

We're now ready to write our formula. Similar to other formula editors in the platform, this formula editor allows you to choose the field and summary type by easily-recognizable labels before inserting the equivalent API values in the editor.

8. Just under the Formula section, select Travel Required from the drop-down list.
9. Select Sum from the next drop-down list.
10. Click **Insert**.

The formula editor now displays the following API representation of those values:

```
Position__c.Travel_Required__c:SUM
```

11. Click the **Insert Operator** drop-down button, and choose **Divide**.
12. Just under the Formula section, select Record Count from the drop-down list.
13. Click **Insert**.

The final formula looks like this:

```
Position__c.Travel_Required__c:SUM  / RowCount
```

We can quickly verify that the formula is correct by checking its syntax before saving.

14. Click **Check Syntax**.
15. Click **Done**, and then click **Next** back in the report wizard.

Try It Out: Adding Columns and Filters

The next three steps allow us to select and order report columns and then define our filters. Because we're already familiar with selecting and ordering columns, let's zip through the first two steps before creating report filters in step six of the wizard.

1. Select the following report columns:
   - `Position: Position Title`
   - `Days Open`
   - `Functional Area`
   - `Status`
   - `Travel Required`
2. Click **Next**.
3. In the Report Column Order area, reorganize the order of the columns as you see fit.
4. Click **Next**.

For our report, we want to define three filters: one to include all positions, one to include only those positions that were created in the last year, and one to include those with a `Status` of Open - Approved or Closed - Filled.

5. In the `View` drop-down list, choose All Positions.
6. In the `Columns` drop-down list, choose Position: Created Date.

Notice that all other date fields defined on the Position object are also available for creating time-dependent filters, including `Close Date`, `Hire By`, and `Open Date`.

7. In the `Duration` drop-down list, choose Current and Previous CY (meaning this and last calendar year).
8. In the first row of the Advanced Filters area, define a filter for "Status equals Open - Approved, Closed - Filled."

9. Click **Next**.

Try It Out: Adding a Chart and Conditional Highlighting

We're almost done. Now, in addition to creating a horizontal bar chart that shows the average number of days open, recruiters, and whether the position requires travel, we also want to define some conditional highlights to help us quickly analyze which recruiters are performing well and which need to work on filling their positions faster.

1. In the `Chart Type` drop-down list, choose Vertical Column - Grouped.
2. In the `X-Axis` drop-down list, choose Average of Days Open.
3. In the `Y-Axis` drop-down list, choose Position: Owner Name.
4. In the `Groupings` drop-down list, choose Travel Required.
5. In the `Chart Title` field, enter Avg Days to Hire With and Without Travel.

In the Conditional Highlighting section, all three of our summary fields are available to highlight, but we just want to emphasize one: `Average Days Open`. That's because we want to highlight which recruiters are closing positions in less than 30 days, less than 60 days, or more than 60 days.

6. In the first row, set the `Summary Totals` drop-down list to Average Days Open.
7. Set the `Low Color` to a shade of green by clicking on the color picker icon and selecting a green color from the popup window.
8. In the `Low Breakpoint` field, enter 30.
9. In the `High Breakpoint` field, enter 60.
10. Set the `High Color` to a shade of red.
11. Click **Run Report**.

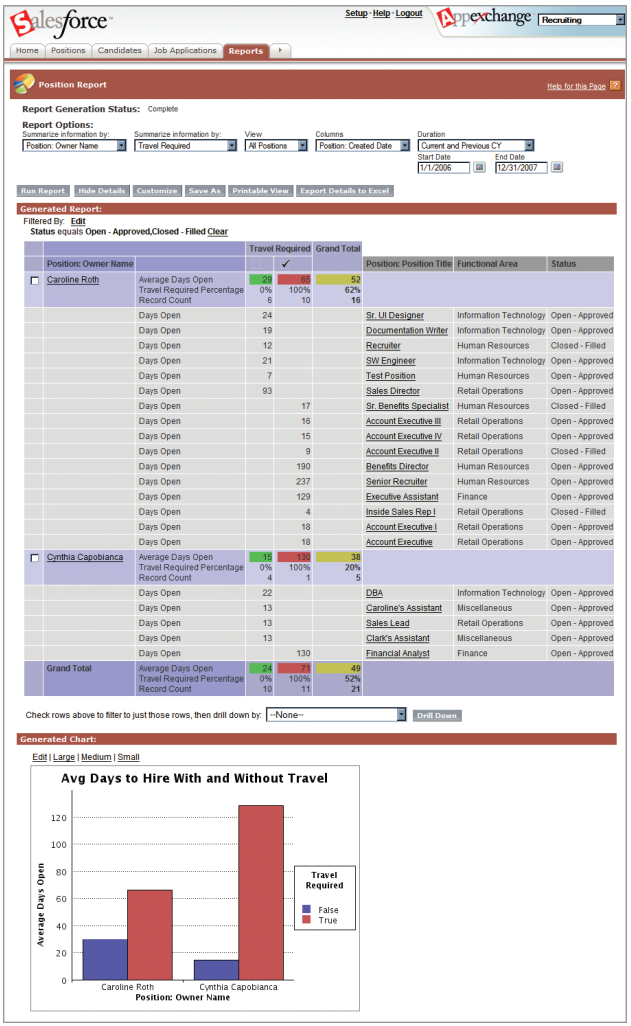


Figure 92: The Final "Avg Days to Hire With and Without Travel" Matrix Report

Ta-da! Our report succeeds in showing how well our recruiters fill positions and provides easy-to-understand insight into why the average number of days for all positions is in the yellow zone—in the report shown in the previous screenshot, the conditional highlighting shows that at Universal Containers, positions that require travel are much harder to fill. Let's quickly save this report now before moving on.

12. Near the top of the report, click **Save As**.
13. In the `Report Name` field, enter Avg Days to Hire With and Without Travel.
14. In the `Report Description` field, enter "On average, how many days does it take each recruiter to fill a position with or without required travel?"

15. From the `Report Folder` drop-down list, select Recruiting Reports.
16. Click **Save**.

As we've seen, custom reports can provide a lot of interesting data that give insight into the challenges that an organization faces. However, unless a user visits these reports on a regular basis, much of their benefit remains untapped. How can we give users a way of keeping tabs on the information in reports without wasting their time? The answer, as we'll see next, lies with *dashboards*.

Introducing Dashboards

A dashboard is a group of different summary or matrix report charts that graphically display custom report data. We can select up to 20 different custom reports to display in each dashboard, and we can arrange them in two- or three-column layouts. Users can browse through all of the dashboards that are available in the Company Dashboards folder in their organization and can also select a favorite dashboard that always displays on the Home tab when logging in. To put it mildly, users *love* the summarized view they get with dashboards, and no good Force.com app is complete without at least one.

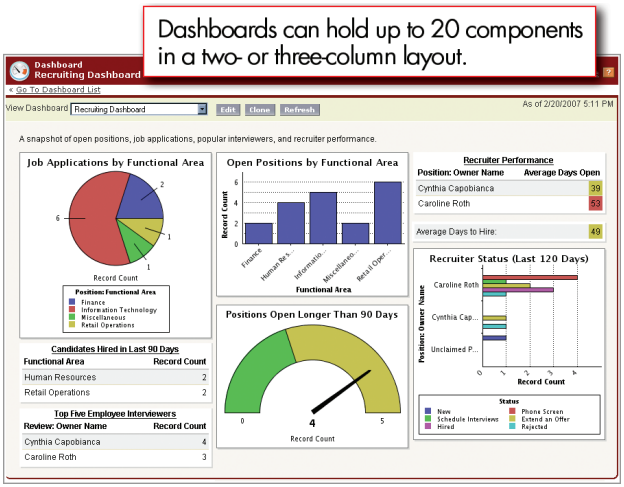


Figure 93: A Sample Recruiting Dashboard

Try It Out: Creating Additional Reports

We'll be creating a small-scale recruiting dashboard in this chapter, but before we do, let's use our new custom report building skills to create a few more reports. Because we're already

familiar with how the custom report wizard works, we'll just outline the specifics of a few more reports in the following table—you can either create them on your own, or just browse through the options that were selected to become familiar with what they are.

Tip: Want to click along with our dashboard instructions but don't want to spend your time creating all these new reports? Just create the Positions Open Longer Than 90 Days report in the first row. However, if you're interested in recreating the sample dashboard displayed here, you'll need to create the other four as well.

Table 38: Additional Recruiting Report Specifications

Report Name	Question	Object/Relationship	Options
Positions Open Longer Than 90 Days	*Which positions have been open for more than 90 days?*	Positions	Type: Summary Report Summarize information by: Location and then by Functional Area Columns: Position: Position Title, Position: Owner Name, Status, Open Date, Hire By, Days Open Filters: View All Positions; Days Open greater or equal 90 Chart Type: Pie Chart Wedges: Functional Area Chart Title: Positions Open Longer Than 90 Days
Job Applications by Functional Area	*Which positions are getting the most candidates?*	Job Applications with Position	Type: Summary Report Summarize information by: Position: Functional Area and then by Position: Position Title Columns: Job Application Number, Status Filters: View All Job Applications; Position: Status equals Open - Approved

Report Name	Question	Object/Relationship	Options
			Chart Type: Vertical Column X-Axis: Position: Functional Area Chart Title: Job Applications by Functional Area
Employee Interviewers	*Which employees conduct the most interviews?*	Reviews with Job Application	Type: Summary Report Summarize information by: Review: Owner Name Columns: Review: Owner Name, Review: Created Date, Job Application: Job Application Number, Job Application: Position, Job Application: Status Filters: View All Reviews Chart Type: Pie Chart Title: Employee Interviewers
Recruiter Status	*What does the job application pipeline look like for each recruiter and open position?*	Job Applications with Position	Type: Matrix Report Subtotal Rows by: Position: Owner Name and then by Position: Position Title Subtotal Columns by: Status Columns: Job Application Number, Position: Position Title Filters: View All Job Applications; Job Application: Last Modified Date Last 120 Days; Position: Status equals Open - Approved, Closed - Filled

Report Name	Question	Object/Relationship	Options
Positions Hired in Last 90 Days	*Who have we hired in the last 90 days?*	Positions	Type: Summary Report Summarize information by: Functional Area Columns: Position: Position Title, Position: Owner Name, Hiring Manager, Functional Area, Job Level, Location, Close Date Filters: View All Positions; Close Date Last 90 Days; Status equals Closed - Filled

Try It Out: Creating a Dashboard

Now that we've got a set of reports to reference, we're ready to create a small Recruiting dashboard. To do so, we'll be working in the Dashboards tab. You can either access it by clicking the arrow tab and selecting it from the list of all available tabs, or by adding it to the list of visible tabs in your Recruiting app, as outlined in *Try It Out: Adding the Reports Tab* on page 198.

1. Click the Dashboards tab and navigate to the standard list page by clicking **Go to Dashboard List** near the top of the screen.

Note: Unlike other tabs, opening the Dashboards tab always displays the last dashboard that you viewed. If you've never visited the tab before, it displays a sample dashboard that comes by default with every organization.

2. Click **New Dashboard**.

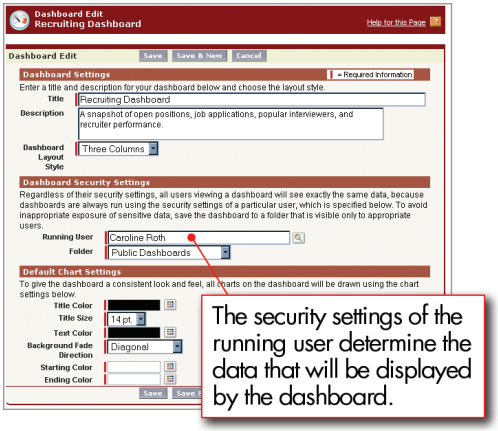


Figure 94: The Dashboard Edit Page

The Dashboard Edit page allows us to define the properties of the dashboard. Once these are specified, we can add report components:

3. In the `Title` field, enter Recruiting Dashboard.
4. In the `Description` field, enter "A snapshot of open positions, job applications, frequent interviewers, and recruiter performance."
5. From the `Dashboard Layout Style` drop-down list, select Three Columns.

The next two fields allow us to define the dashboard's security. Dashboards are always executed with the security settings of a single user, the *running user*. 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.

Note: The running user's security settings only apply to the dashboard view. Once a user drills down into a source report or detail page off the dashboard, the user will view the data based on his or her normal security settings.

For example, suppose a system administrator with the "Modify All Data" permission is the running user for our Recruiting Dashboard. In this case, every recruiting-related record is counted in all of the report totals on our dashboard, including users who'd normally be restricted from viewing certain records (like those assigned to the Standard Employee profile). Although those users would be able to see the summary data for all records in the dashboard, if they ever navigated to the source reports, they'd only see the records they have permission to view.

Consequently, when designing a dashboard it's important to be aware of the audience of the dashboard and how much data they should be able to see. If you do give a user access to dashboards that include more data than he or she normally has permission to view, be sure to

communicate that they might see different values when they click through the dashboard to view the associated reports. And if you need to restrict a dashboard from certain users, just save it to a restricted-access folder.

For our Recruiting Dashboard, the data that we'll be showing in the dashboard isn't particularly sensitive. Consequently, we'll choose a system administrator as the running user, and save the dashboard to a public folder.

6. In the `Running User` lookup, select a user with system administrator privileges.
7. In the `Folder` drop-down list, select Company Dashboards.

Other settings on this page allow you to customize the look and feel of the dashboard colors and fonts. We'll leave these as the defaults for now.

8. Click **Save**.

We now have an empty dashboard that's ready to be filled with dashboard components.

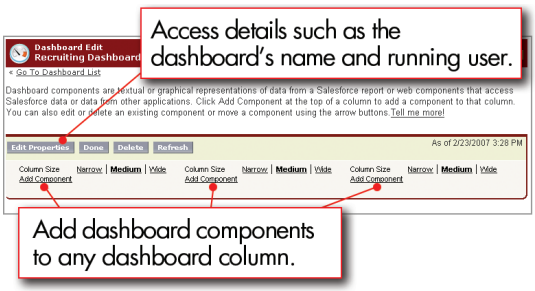


Figure 95: The Empty Recruiting Dashboard

Adding Dashboard Components

The Dashboard Edit page now shows us the contents of the dashboard itself. Although we have no components yet defined, we can add them to any of the three columns that are displayed, and we can move them around in any order, regardless of the order in which we define them. We're allowed up to 20 components per dashboard, but only the top row of components care visible in a user's Home tab when he or she chooses a dashboard to display on the Home tab.

Components come in five varieties:

- *Charts*—Displays a pie chart, bar chart, line chart, or any other type of chart that can be made in a report.

- *Tables*—Displays a two-column table that contains record counts and values from the top-level row grouping in the report.
- *Metrics*—Inserts the grand total of a report at the end of a label that you customize.
- *Gauges*—Uses the grand total of a report as a point on a scale.
- *Custom S-Controls*—Displays any custom content that can be viewed in a Web browser, such as an ActiveX control, an Excel file, or a custom HTML Web form.

We'll talk about the first four component types in this chapter, and then you'll learn more about custom s-controls in the next chapter on composite applications.

Try It Out: Adding a Chart Component

Let's start by adding a chart that shows the number of open positions by functional area:

1. In the left column, click **Add Component**.
2. Select `Chart` as the component type.
3. In the `Title` field, enter Open Positions by Functional Area.
4. From the `Custom Report` drop-down list, select Open Positions by Functional Area.

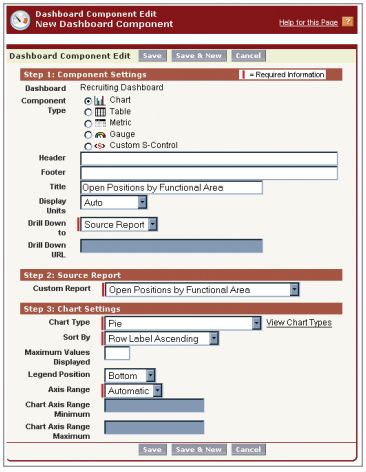


Figure 96: The Dashboard Component Edit Page

We're now ready to enter specifics about how our chart should be formatted. Notice that even though we already defined a chart for the Open Positions by Functional Area report, the corresponding dashboard component needs to have the chart element redefined. That's because charts that are useful in a report aren't always as useful in the smaller scale of a dashboard:

5. From the `Chart Type` drop-down list, select Pie.
6. Click **Save**.

Ta-da! We now have a pie chart in our dashboard! If a user clicks on the chart, they're taken to the report from which the chart was generated.

Try It Out: Adding a Gauge Component

Now let's create a gauge component that shows us how we're doing with positions that have been open for longer than 90 days:

1. In the center column, click **Add Component**.
2. Select `Gauge` as the component type.
3. In the `Title` field, enter Positions Open Longer Than 90 Days.
4. From the `Custom Report` drop-down list, select Positions Open Longer Than 90 Days.

Now on to the appearance of our gauge. Just like the conditional highlighting in our matrix report, this gauge can display different colors depending on the total count of positions that have been open too long. We want a green color if there are less than two positions, yellow if there are between two and five, and red if the value is over five.

5. In the `Minimum Value` field, enter 0.
6. From the `Low Range Color` color picker, choose a shade of green.
7. In the `Breakpoint #1 Value` field, enter 2.
8. In the `Breakpoint #2 Value` field, enter 5.
9. From the `High Range Color` color picker, choose a shade of red.
10. Click **Save**.

Perfect! Our Recruiting Dashboard now contains two components.

Try It Out: Adding a Table Component

Now let's create a table component that shows us the average number of days it takes each recruiter to fill a position:

1. In the right column, click **Add Component**.
2. Select `Table` as the component type.
3. In the `Title` field, enter Recruiter Performance.
4. From the `Custom Report` drop-down list, select Avg Days to Hire With and Without Travel.

Just like the corresponding report, we can add conditional highlighting to our table to highlight recruiter performance levels. We want a green color if the recruiter needs less than 45 days, yellow if he or she needs between 45 and 75 days, and red if the value is over 75.

5. From the `Low Range Color` color picker, choose a shade of green.
6. In the `Breakpoint #1 Value` field, enter 45.
7. In the `Breakpoint #2 Value` field, enter 75.
8. From the `High Range Color` color picker, choose a shade of red.
9. Click **Save**.

Try It Out: Adding a Metric Component

Finally, let's create a metric component that shows us the average number of days it takes all recruiters to fill a position:

1. In the right column, click **Add Component**.
2. Select `Metric` as the component type.
3. From the `Custom Report` drop-down list, select Avg Days to Hire With and Without Travel.

Because metric components consist of a single grand total value, they don't need a title. Instead, we give them a label much like any other field that you see in the platform:

4. In the `Metric Label` field, enter Average Days to Hire.

Note: Did you notice that we're using the same report that we used for our table? This metric is just another interpretation of the data.

Finally, let's add more conditional highlighting to showcase values:

5. From the `Low Range Color` color picker, choose a shade of green.
6. In the `Breakpoint #1 Value` field, enter 45.
7. In the `Breakpoint #2 Value` field, enter 75.
8. From the `High Range Color` color picker, choose a shade of red.
9. Click **Save**.
10. Click **Done**.

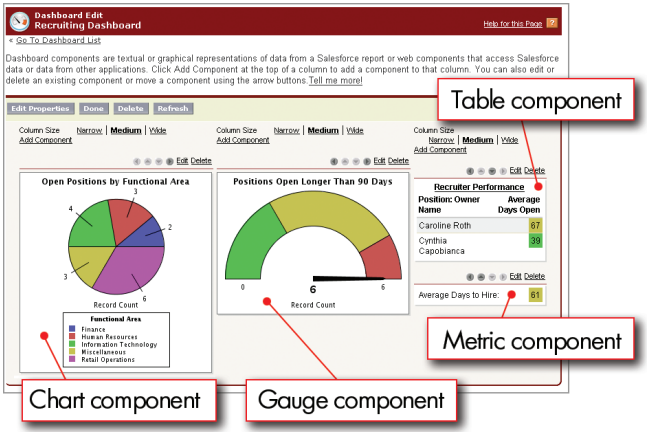


Figure 97: Four Components on the Recruiting Dashboard

We've now got a simple, four-component dashboard for our Recruiting app! If you click the Home tab, you can add the dashboard's first row to your home page:

11. On the Home tab, click **Customize Page** in the upper right corner of the Dashboard component.
12. Select Recruiting Dashboard from the `Dashboard Snapshot` drop-down list.
13. Click **Save**.

Ta-da! And because we saved the dashboard in a public folder, any user can add it to his or her Home tab, too.

As you can see, once we'd created the reports for our Recruiting app, adding them to a dashboard was a piece of cake. It's so easy, that we'll leave the remainder of the components as an exercise for you to try on your own—see how close you can come to recreating *Figure 93: A Sample Recruiting Dashboard* on page 214.

Look At What We've Done

Check out our Recruiting app now! We've almost met all of the requirements we talked about back at the beginning of the book, but we're not done yet. In the next chapter, we're going to move beyond the native capabilities of the platform, like security and workflow, and introduce s-controls and the Force.com Web Services API. This API provides the key to incorporating functionality from all over the Web and will help us create a truly powerful on-demand application. In fact, once you've mastered the tools available in the platform, the API will be the way that you can let your creativity soar—the functionality of any app you build on the platform will be limited only by the Web itself!

Chapter 10

Moving Beyond Native Apps

In this chapter ...

Up to this point, we've built a compelling app using various parts of the platform. We've created a data model to store our recruiting information, put workflow and approval logic in place to help manage the data, and built reports and a dashboard to help share the data.

Because of the power of the platform, we were quickly able to put all of these pieces together in the user interface, without any traditional coding. However, if we revisit the requirements of our Recruiting app, we'll see there's still another item to implement, and it's not quite as straightforward.

We need to create a mapping feature that shows where each candidate is located. Clearly no such functionality is native to the platform, and we certainly don't have time to build our own mapping engine. Is this type of functionality even possible?

Of course it's possible! Because the Force.com platform runs on the Web, we can leverage the power of other websites to implement features that would never be available just through our platform alone. We can use our HTML and JavaScript skills to mash up Web mapping services from Yahoo! with our own recruiting data, and then place this functionality in our own app.

To implement this kind of composite application feature, we'll first need to learn about the technology that makes it possible: Web services in general and the Force.com Web Services API in particular. Once we review those technologies, we'll implement the candidate map and see

how easy it is to create a rich, user-centric app with few (if any!) boundaries.

Tip: To get the most out of this chapter, you should be familiar with HTML, JavaScript, and server-side scripting. However, even if you're not an experienced developer, you can still follow along to see what can be done with the platform.

Introducing Web Services

First we need to look at the technology that makes composite application pieces like our candidate mapping feature even possible: Web services. A *Web service* is the mechanism by which two applications that run on different platforms, that were written in different languages, and that are geographically remote from each other, can exchange data using the Internet. Web services makes data exchange between two such applications as straightforward as the way that two processes can exchange data on a single computer.

The way that data is exchanged between two Web services is similar to the way that data that is exchanged between a Web browser like Microsoft Internet Explorer and a Web server. Just as a Web browser uses a common network protocol (HTTP over TCIP/IP) to download HTML files hosted on a Web server, a Web service can also use this same network protocol to download data from another Web service. The key difference is the actual data that is sent and received—Web services use XML instead of HTML.

So why are we talking about Web services? Well, it turns out that the platform includes a Web services API that allows us to share data with other Web services on the Internet. Couple our Web services API with a tool like Yahoo! Maps, and we're well on our way to implementing our candidate mapping feature! Before we jump in, though, let's see how the API works.

Introducing the Force.com Web Services API

The API defines a Web service that enables full, reliable access to all of the data in our organization, including the ability to read, create, update, and delete records. To get a better idea of what we need to do to use the API, though, we need to discuss its most common calls, how it acts as a Web service, and how to authenticate API calls so that our data remains secure.

Common API Calls

In addition to the basic read, write, update, and delete calls, the API also allows us to request metadata related to our standard and custom objects, maintain passwords, perform searches, and much more. Although the API supports more than 20 different calls, there are just four key calls that make up the majority of all API traffic, as shown in the following table:

Table 39: Commonly Used API Calls

Call	Description
query	Returns data according to a SQL-like syntax called SOQL (Salesforce Object Query Language) SOQL specifies what kind data we're looking for and the criteria that should be used to select the data. For example, to retrieve the `Id` and `Name` field values for each account whose name ends with Tool, we'd use: `(Select Id, Name From Account Where Name Like '%Tools').`
create	Defines a new record for a particular object.
update	Edits an existing record according to its record ID.
delete	Erases a record according to its record ID.

Although the exact use and syntax of API calls is beyond the scope of this book, you can find detailed documentation and API coding examples on the Apex Developer Network at www.salesforce.com/developer/.

SOAP, XML, and the API

When we talked about Web services previously, we learned that they communicate with one another by passing XML data over the Internet. Fortunately, that doesn't mean that we have to brush off our XML code references and expect to bury ourselves in DTD files when we create our composite application component. It turns out that the API uses SOAP (Simple Object Access Protocol), a protocol that defines a uniform way of passing XML-encoded data. This is great for developers like us, since virtually every major programming language has a built-in ability to create and consume SOAP messages. As a result, all of the details about handling Web services XML will be hidden by familiar-looking objects in whatever language we choose for coding. That means we'll never need to actually write or read any XML ourselves when we're working on our app.

Authenticating Calls to the API

Finally, let's talk about authentication. Some Web services are public, meaning that any request to the Web service will be honored and the request carried out. This makes the lives of

developers easier, since they don't have to jump through any hoops in order to access the data that the services provide, but it also means that anyone can access the data! With the private, valuable data of your organization stored on the platform, that's just unacceptable. Consequently, the API is a private Web service, requiring authentication to access the data, just as a user of your Recruiting app is required to log in to access any data.

The API provides authentication through the use of a *session ID*, a short-lived token that provides access to all API calls. This token is analogous to a driver's license—when you're issued a driver's license, it expires after a period of time, but until it expires you're authorized to drive an automobile. Likewise, when you're issued a session ID, it expires after a period of time (a configurable period of anywhere from 30 minutes to eight hours), but until it expires you're authorized to make Web service calls using the API.

There are two ways of acquiring a session ID:

- Using the `login()` call (best for composite application pieces that originate from outside of the Salesforce user interface)
- Using merge fields (best for composite application pieces that reside inside the Force.com platform user interface)

Authenticating with the login() Call

The first way to make authenticated Web service calls to the API is by using the `login()` call. Similar to the way the main user login page works in the user interface, the `login()` call takes a username and password and executes a login sequence on `https://www.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 hosts (for example, na1.salesforce.com, na2.salesforce.com, and so on), 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.

This method of authentication is appropriate for composite components that originate from outside of the Salesforce user interface. For example, if we were to write an external Web page that accepts candidate resumés for different open positions, we could use the API's `login()` call to upload these candidates and resumés to our Recruiting app.

Figure 98: Authenticating with the login() Call

Authenticating with Merge Fields

The second way you can make authenticated Web service calls to the API is by using merge fields. A *merge field* is text that gets replaced with a value when the platform delivers a Web page to a user's browser, similar to the way Microsoft Word generates labels or form letters using a database of addresses. We'll talk about merge fields more when we get to writing the code for our candidate mapping tool. For now, you just need to understand that the platform provides the user's session ID and that we can obtain this session ID using a merge field. That means our composite application components can acquire the session ID of a user who's already logged in to the system without forcing him or her to log in again.

Note: The merge field method is also nice because it tells us about the user who's currently logged in. When our composite application piece uses this information to make API calls, the API returns only data to which the user has access, according to his or her security and sharing settings.

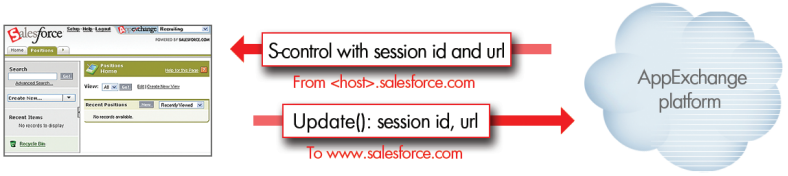


Figure 99: Authenticating with Merge Fields

So once you've obtained a session ID and server URL, what do you do with them? The answer to this question depends on what programming language you're using to create your composite application piece. In general terms, you'll include the session ID in every API call, and you'll direct your composite application piece to make the API request to the server URL that you obtained. We'll see examples when we finally get to coding our candidate mapping feature.

Implementing a Composite Application Piece: The Candidate Mapping Tool

All right! Now that we've reviewed Web services and the API, let's get started implementing the composite application piece for our Recruiting app: the Candidate Mapping tool. Our requirements state that we need to generate a map for all the candidates that have applied for a particular position so that we can better understand potential relocation costs associated with a new hire. Since Universal Containers has offices all over the country, this map will also help us assign candidates to an office if a particular position is open in more than one location.

We know we can create this tool by mashing up the Force.com API and Yahoo! Maps with some basic JavaScript and HTML. But where do we put this code, and how do we incorporate it into our app?

Introducing S-Controls

It turns out that the platform includes a particular component that gives us both a place to store our composite application piece code and an easy way of adding it to any page layout or tab. This component is called an s-control.

An *s-control* is essentially an instruction that defines what the platform should render on a page. S-controls can contain any type of content that you can display or run in a browser, such as a Java applet, an ActiveX control, a Microsoft Excel file, or a custom HTML Web form. In addition to any file you upload, a custom s-control contains HTML code that controls how the content is displayed and referenced in a custom link. S-controls provide a flexible, open means of extending the platform interface beyond what's currently available.

S-controls come in three flavors: Custom HTML, URL, and Snippet:

HTML S-Controls
: 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.

URL S-Controls
: 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 ultimately hosted by an external website.

Snippet S-Controls
: An s-control that contains HTML or JavaScript that is meant 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 code that can be used in multiple s-controls.

For our Candidate Mapping tool, let's create an HTML s-control that contains standard HTML and JavaScript. A combination of these two technologies actually gives us a lot of power, especially when you consider all of the work being done lately with AJAX (a Web development technique for creating interactive Web applications).

Try It Out: Defining the Candidate Mapping S-Control

Let's go ahead and create our candidate mapping s-control. Once it's defined, we'll hook it in to our Recruiting app's user interface and play around with it to make sure it works:

Note: There's a lot to talk about in this exercise, so we're going to break it up in a couple of sections to help us organize our thoughts.

1. Click **Setup ➤ Build ➤ Custom S-Controls**.
2. Click **New Custom S-Control**.

Try It Out: Setting S-Control Attributes

Upon clicking **New Custom S-Control**, we get to the Custom S-Control Edit page.

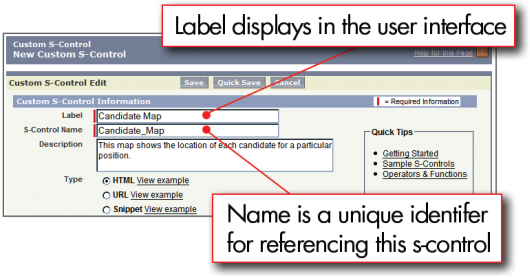


Figure 100: S-Control Edit Page—Attribute Entry

The top of this page consists of three text boxes that allow us to enter a label, name, and a description of our s-control. The `Label` field contains the name for the s-control that will be displayed in various lists and page layouts. The `Name` field specifies the name that other s-controls can use to reference this s-control (particularly useful when using snippets), and the `Description` field provides a place to briefly indicate what the s-control does. (Other users will be very thankful if you provide good descriptions for your s-controls, because they're often hard to decipher at a single glance.)

3. In the `Label` field, enter Candidate Map.
4. In the `S-Control Name` field, accept the defaulted value of Candidate_Map.
5. In the `Description` field, enter "This map shows the location of each candidate for a particular position."
6. Select the `HTML` type.

Try It Out: Entering Our Code into the S-Control Content Area

Below the s-control attribute area is the content area for our s-control—this is where we'll enter the HTML and JavaScript code that's going to implement our mapping tool. You'll find that it's very similar to the formula editors that we've seen before—you can insert merge fields, operators, and functions, and you can even check syntax when you're done.

The code block that we're using provides a simple implementation of our candidate mapping tool. It combines candidate address information with Yahoo! Maps to display those candidates who have applied for a particular position. This isn't really a coding book, but in the next few sections we'll briefly examine a few lines of the code that are related to the API (the bolded sections) to see how we've leveraged some of the concepts we talked about earlier in the chapter. Now let's go ahead and enter our code into the s-control:

7. Enter the code block that appears below into the s-control's large text area box.
8. Click **Check Syntax** to verify that you haven't made any typos.
9. Click **Save**.

Tip: Alternately, download a copy of the code from http://wiki.apexdevnet.com/index.php/Creating_On-Demand_Apps and save it on your computer. (A file containing the code was included in the zip file you may have downloaded with the CSV import files in *Chapter 6: Expanding the Simple App Using Relationships* on page 81.) Open the code in your favorite text editor. Select all the lines of code, and then copy and paste them into the content area.

```
<html>
<head>
<style type="text/css">
#mapContainer {
height: 475px;;
width: 600px;
}
</style>

<script language="javascript" src="/soap/ajax/8.0/connection.js"
        type="text/javascript"></script>
<script type="text/javascript"
```

```
src="http://api.maps.yahoo.com/ajaxymap?v=3.0&appid=salesforce.com">
        </script>

<script type="text/javascript">
var conn;
function initPage() {
   conn = sforce.connection;
   loadMap();
}

var reload = true;
var map;
var link;
var geoPoints = [];
var dataPointCount = 0;
var pointCounter = 0;

function geoCodeEnd(r) {
    if (r.success == 1) {
        if (r.GeoPoint.Lat != 0) {
            geoPoints[geoPoints.length] = r.GeoPoint;
            pointCounter++;
        }
    } else {
     pointCounter++;
    }
    if (pointCounter >= dataPointCount) {
        var bestZoomLevel = map.getZoomLevel(geoPoints);
        if (bestZoomLevel < 5) bestZoomLevel = 5;
        var geoCenter = map.getCenterGeoPoint(geoPoints);
        map.drawZoomAndCenter(geoCenter, bestZoomLevel);
        map.panToLatLon(geoCenter);
    }
}

function loadMap() {
    // Create a map object
    map = new YMap(document.getElementById('mapContainer'));
    // Display the map centered on given address
    YEvent.Capture(map, EventsList.onEndGeoCode, geoCodeEnd);
     map.addZoomLong();

    doQuery();
}

function doQuery() {
    if (reload == false) return;
    var positionId = "{!Position__c.Id}";
    var qr = conn.query("Select Candidate__c From Job_Application__c
                         Where Position__c = '" + positionId + "'");

    if (qr.size > 0) {
        var ids = [];
        var records = qr.getArray("records");
        for (var i=0;i<records.length;i++) {
            ids[i] = records[i].get("Candidate__c");
```

```
        }
        conn.retrieve("Id, Last_Name__c, First_Name__c, Street__c,
                       City__c, State_Province__c,
                       Current_Employer__c", "Candidate__c",
                      ids, plotMarkers);
    } else {
        // Display map of US if no Candidates found
        var myPoint = new YGeoPoint(40,-95);
        map.drawZoomAndCenter(myPoint, 14);
        alert("There are no candidates for this position to map.");
    }
    reload = false;
}

function doNav(arg) {
    window.parent.location = "/" + arg;
}

function plotMarkers(qr) {
    var records = qr;
    if (qr.getArray) {
        records = qr.getArray("records");
    }
    var counter = 1;
    if (records.length > 0) {
        for (var x=0;x<records.length;x++) {
            var db = records[x];
            if (db.get("Street__c") != null &&
                      db.get("City__c") != null &&
                      db.get("State_Province__c") != null ) {
                dataPointCount++;
                var addr = db.get("Street__c") + ", " +
                              db.get("City__c") + ", " +
                              db.get("State_Province__c");
                var marker = new YMarker(addr);
                marker.addLabel(counter);
                var name = db.get("First_Name__c") + " " +
                                  db.get("Last_Name__c") ;
                counter++;
                name = "<a href='" +
                       conn.describeSObject("Candidate__c").
                          urlDetail.replace("{ID}", db.get("Id"))
                       + "' target='_top'>" + name + "</a>";
                marker.personName = name;
                YEvent.Capture(marker, EventsList.MouseClick,
                    function() {this.openSmartWindow
                                    ("<div>This is where " +
                                      this.personName +
                                      " lives!<BR> </div>") });
                map.addOverlay(marker);
            }
        }
    }

}
```

```
</script>


</head>

<body onload="initPage();">
<div id="mapContainer"></div>

</body>
</html>
```

Including the AJAX Toolkit Library

In the code sample, several key parts relating to the API have been highlighted in bold font. The first, repeated here, is the bit of HTML needed to include the AJAX Toolkit JavaScript library:

```
<script language="javascript" src="/soap/ajax/8.0/connection.js"
        type="text/javascript"></script>
```

This library is what enables our s-control to interact with our data using the API. Once this line is processed by the browser, every API call that is described in the API documentation is now available to the JavaScript in the s-control. The syntax for using this toolkit is very similar to the syntax provided in the API documentation.

Writing a Function for the Body Tag's Onload Event

Another important concept is the `initPage()` function. Notice that it's called when the page's `onload` event occurs:

```
var conn;
function initPage() {
   conn = sforce.connection;
   loadMap();
}
```

Whenever we're writing an s-control, it's very important to write a function that handles this event in order to ensure that our s-control's HTML elements have been completely sent to the browser before we run any JavaScript. This is necessary because Web browsers render the HTML received from a Web server serially, but they also start processing the JavaScript that has been included in an HTML page as soon as they can. This means that JavaScript may begin to run before all of the HTML elements have been sent to the browser. By starting our JavaScript execution when the body tag's `onload` event fires, we can rest assured that any elements we want to access with JavaScript will already be represented in the browser.

Using Merge Fields to Set the Session ID

The next bit of highlighted code demonstrates a key concept: merge fields. Merge fields are the mechanism by which we can obtain specific and contextual information for our s-control code.

```
var positionId = "{!Position__c.Id}";
```

In this case, the Position ID is made known to the executing JavaScript code through the use of the `{!Position.Id}` merge field. The ID field is common to every standard or custom object record, and uniquely identifies the record in the platform. We can use this ID value in an API query to limit the results to only those records that contain that ID. In this line of code, we're grabbing the position record that matches this ID so that we can obtain its related candidates for mapping.

In addition to the ID, we could have requested just about any and every other field of the Position object using the merge fields technique.

We can now use this value to create a SOQL statement that specifies what data we'd like to pull out of our app. Using the AJAX toolkit we don't need to worry about authentication. When the toolkit is sent down to the browser, the important authentication data required, `session id` and `server url`, are automatically merged into the toolkit code.

As you can see, merge fields represent a very powerful mechanism for requesting data from the platform. Many s-controls use merge fields exclusively and forego the need to use the API to request data altogether. The only drawbacks to merge fields are that they're always read only, and they can't cause data to be created or updated on the platform.

Making an API Call

The next highlighted segment is an actual API call (remember `conn = sforce.connection;` appeared earlier in the code):

```
var qr = conn.query("Select Candidate__c From Job_Application__c
                        Where Position__c = '" + positionId + "'");
```

This query call is asking for all the Candidate record IDs in the Job Application records whose Position IDs match the one given to us in the merge field. Once the method executes, we'll know which Candidate records we need to map.

Accessing Data

The last two highlighted snippets demonstrate how to access the data that resulted from our query:

```
if (db.get("Street__c") != null &&
             db.get("City__c") != null &&
             db.get("State_Province__c") != null ) {
     dataPointCount++;
     var addr = db.get("Street__c") + ", " + db.get("City__c") +
                          ", " + db.get("State_Province__c");
     var marker = new YMarker(addr);
     marker.addLabel(counter);
     var name = db.get("First_Name__c") + " " +
                   db.get("Last_Name__c") ;
     ...
}
```

Our query resulted in a set of Candidate records that matched the criteria of the query. To obtain the address value on each of these records, we can loop over the set. In JavaScript, a record is defined as an instance of an *SObject* object, so we'll assign each object in the set of records to a variable called `db`.

SObject objects provide various methods for retrieving and setting values on the record, including both a dot notation and a key notation for accessing field values. The key notation expects a field name as an index and returns the value from the field for that record. An example of the key notation is `db.get["First_Name__c"]` as opposed to the dot notation form, `db.First_Name__c`. Both accessor forms are case sensitive. Notice that there isn't any XML, SOAP, WSDL, or other details of the actual Web service in this function. Instead, we have a simple object from which we access values using simple methods.

This ability to easily retrieve the information from our application is one of the most powerful features of the platform. The custom objects and fields that we created in previous chapters are all instantly available through the API.

Introducing Hooks and Targets

By now you should have a better feel for what s-controls are and how they allow us to create composite application pieces in our app. And if you've been following along really closely, you might also be asking yourself a couple of questions. For instance, how and when will our users see and use the candidate mapping s-control that we just defined? And, how does the platform know what Position ID to send us?

It turns out that we can answer both of those questions by talking about hooks and targets. A *hook* is a location in the user interface from which we can launch or kick off an s-control or

URL—the *target*. Below is a table of all the hooks that we have available to us in the platform, along with the targets that we can launch from those hooks:

Table 40: Hooks and their Possible Targets

Hook	Targets
Custom Link or Button	S-Control, URL
Web Tab	S-Control, URL
Inline S-Control	S-Control

When we defined the custom objects for our Recruiting app, we also defined tabs and page layouts to display those objects. Since our candidate mapping tool shows the locations of candidates for a particular position, we'll hook our s-control to the Position object's page layout with a custom link. The platform will know which Position ID to send us because we'll see our s-control only when we're looking at a particular Position record.

Try It Out: Creating a Custom Link

Now that our custom s-control is written and saved to the platform, we're ready to create the hook on which to hang this code. The first step is to create a custom link.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Custom Buttons and Links related list, click **New**.

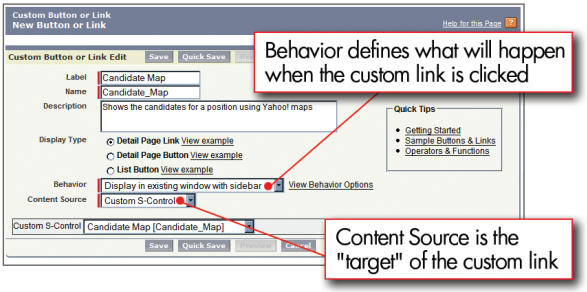


Figure 101: Custom Link Edit Page

The custom link edit page should look familiar, since it closely resembles the s-control editor that we saw previously.

4. In the `Label` text box, enter Candidate Map.
5. In the `Name` text box, accept the defaulted value of Candidate_Map.
6. In the `Description` text box, enter "Shows the candidates for a position using Yahoo! Maps."

Depending on the type of action that we want to take, we can choose how to display our s-control in a Position record's detail page. Actions that don't have any affect on the record (like our Candidate Mapping tool) can be displayed as a link on the record's detail page. Actions that affect the record itself should be displayed as a Detail Page Button, and actions that affect multiple Position records in a list should be displayed as a List Button.

7. Select `Detail Page Link` as the type.

Next choose the action that we want to perform: we can either open a new window, display in the existing page (with or without sidebar and header), or simply execute some JavaScript. Because we want our candidate map to appear as part of our Recruiting app, let's choose Display in existing window with sidebar.

8. In the `Behavior` drop-down list, select Display in existing window with sidebar.

Because we implemented our Candidate Mapping tool with an s-control, the content source, or target as we called it, should be Custom S-Control.

9. In the `Content Source` drop-down list, select Custom S-Control.

Selecting Custom S-Control changes the layout of the custom link edit page so that instead of a large text area box we see only a `Custom S-Control` drop-down list.

10. In the `Custom S-Control` drop-down list, select Candidate Map [Candidate_Map].

Notice that our s-control is listed in this drop-down list according to both its label and internal name.

11. Click **Save**.

After clicking **Save**, we are reminded that no users will be able to see the link until we add it to a Position page layout. Makes sense, right? Creating a custom link is similar to adding a custom field to an object—it's defined in the database, but no user will be able to see it until we explicitly put it somewhere.

Try It Out: Adding a Custom Link to a Page Layout

So let's do it! All we need to do to finish implementing our Candidate Mapping tool is add it to our position page layout.

1. Click **Setup ➤ Build ➤ Custom Objects**.
2. Click **Position**.
3. In the Page Layouts related list, click **Edit** next to Position Layout.

By now this should be a familiar page to you!

4. In the `View` drop-down list on the right side of the page, select Position Custom Links.

This action causes the table below the drop-down list to be filled with all of the custom links that have been created for the Position object. Since we've created only one so far, that's all we'll see in this list.

5. Click and drag the Candidate Map custom link over to the Custom Links area of the page layout.

Unlike regular fields, custom links can be placed only in the special Custom Links layout section.

6. Click **Save**.

We did it! Although the coding of the s-control was a little tricky, every other step was just as straightforward as what we've already done in previous chapters. Let's now go try it out!

Try It Out: Running Our S-Control

To try out our new candidate mapping tool, all we need to do is visit a Position record.

1. Click the Positions tab.
2. Select the Sr. Benefits Specialist record.
3. In the Custom Links section, click **Candidate Map**.

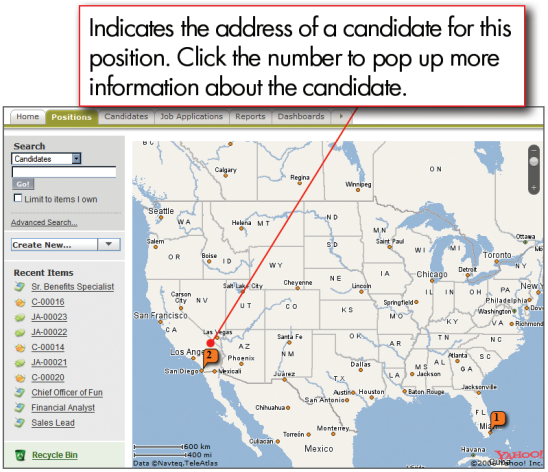


Figure 102: Our Working Candidate Mapping Tool

Voilà! Notice how our s-control is embedded within our Recruiting app as if it were natively built. Also notice how we didn't need to do a special login just for our s-control, and that it could automatically figure out which Candidate records to display based on the Position record we were just visiting. Pretty cool!

Once again, welcome to the power and flexibility of the platform. With fewer than a hundred lines of code, we've introduced non-native functionality that looks as if it were made expressly for our Recruiting app. Couple the Force.com Web Services API with the thousands of Web services that are available on the Internet today, and the possibilities are limited only by our imaginations!

Chapter 11

Learning More

In this chapter ...

This book has introduced some of the native technologies associated with the Force.com platform. We've created a fully functional Recruiting app, and we've introduced s-controls and the Force.com Web Services API to show you how it can be used to build composite apps. But there's only so much we can cover in a single book—we skipped over many other powerful tools and options, and didn't talk about how you can share your own apps with others on the AppExchange. Indeed, we'd be surprised if you didn't have more questions about how you can take advantage of all that the platform has to offer!

Fortunately there are a number of ways you can learn more about the platform.

The Apex Developer Network

The Apex Developer Network (ADN) website is the place to be for Force.com developers. Available at www.salesforce.com/developer/, it provides a full range of resources including sample code, toolkits, an online developer community, and the test environments necessary for building apps. It includes an online version of this book (written by ADN staff members) and has information about the ADN@Dreamforce event that we hold every year for Force.com developers. If you need more info, have a question to ask, are seeking a toolkit or sample, or just want to dig a little deeper into Force.com development, the ADN website is the first place you should go.

Let's take a quick tour:

- Get a Developer Edition account.

  Take your application-building skills to the next level. A Developer Edition account is a free, fully-functional Salesforce account that allows you to get hands-on experience with all aspects of the platform in an environment designed for development. If you haven't done so already, go to the ADN home page and create a Developer Edition account for yourself. Then use it to try out new application ideas and to test various Force.com tools and technologies.

- Get the resources you need on the ADN wiki.

  Of course, a big part of ADN's mission is to ensure that you and every other Force.com developer has the support necessary to build great apps. To this end, you'll want to know about the Wiki section of the ADN site—http://wiki.apexdevnet.com/index.php/Apex_Wiki. This is the place to turn for tools, documentation, and reference information for developers, whether you're building native apps like the Recruiting app, composite apps that use the Force.com Web Services API, writing Apex, or getting ready to distribute your app on the AppExchange. And because it's all maintained on a wiki, you'll be able to see tips, tools, and best practices from other ADN members, as well as contribute your own! By developers and for developers, the wiki is the heart of the ADN website.

- Get the next volume in the Force.com book series—the *Force.com Cookbook*

  This book moves beyond the native functionality of the platform to give you an in-depth look at developing with Force.com. It provides over 100 "recipes" for writing s-controls, using the Force.com API, developing Apex scripts, and creating Visualforce pages. Get it on the ADN website at wiki.apexdevnet.com/index.php/Platform_Cookbook.

- Keep up with the ADN community.

Your fellow developers of all levels of sophistication are maintaining lively conversations every day on the ADN discussion boards at community.salesforce.com/sforce?category.id=developers. You're welcome to join in by posting questions and comments of your own, or you can just read along and learn. The free monthly ADN newsletter provides a recap of the most interesting news and information, delivered right to your email inbox.

- Get the latest Force.com news.

  The ADN blog at http://blog.sforce.com/ is where the ADN team makes announcements and shares some of what it has learned in the process of creating and managing more than 300 apps on the AppExchange directory. This blog thrives on feedback and the cross-pollination of ideas. We welcome your comments.

- Take part in ADN events and activities.

  ADN also sponsors various special events and activities, like the very popular ADN Seminars tour, training classes, and our biggest event of the year, ADN@Dreamforce—the conference within a conference for Force.com application developers. Information on upcoming events is posted on the ADN site at www.salesforce.com/developer/ and is included in the ADN newsletter.

Help and Training Options

In addition to the ADN website, the platform itself offers lots of help and training options:

- **Find Answers to Your Questions**

  Click **Help** at the top of any page in the application. Enter your keywords in the Search box and click **Go!**. The search returns online help topics, knowledge base solutions, and recommended training classes that match the keywords you entered.

- **Take Training**

  Select the Training tab of the Help & Training window, choose your role and geographic location, and click **View Classes!** to find free online training classes. All kinds of training is available to ensure your success and application expertise:

  - Classroom training

    Gain the knowledge and skills you need through instructor-led workshops and courses that provide comprehensive, intensive hands-on sessions, including Salesforce: Administrator Essentials, Force.com: Building Applications, and Force.com: Migrating

Data. All classes are taught by expert instructors, include customized and comprehensive training materials, and are offered in many convenient locations.

- Custom training

  Salesforce.com can also bring the classroom to your organization.

- Online learning

  Take advantage of more than fifteen online training classes, available to you on-demand, 24/7, and just few clicks away!

To learn more or register for training, contact Successforce Education Services at EducationServices@salesforce.com, or visit www.salesforce.com/services-training/education-services/.

- **Download Tip Sheets and Best Practice Guides**

  Select the Help tab of the Help & Training window and click **Tips** in the task bar to view and download tip sheets, implementation guides, and best practices for specific features.

Podcasts

Thanks to the hard work of our colleagues on the Successforce website, we have an impressive number of podcasts available on our iTunes channel. You can access our iTunes area by searching for "salesforce" on the iTunes Music Store.

This public podcast channel keeps the Force.com community connected by providing free access to a wide range of best practices, case studies, and product- and platform-focused digital audio content via the iTunes Music Store. Access to expert Force.com and CRM content is a quick download away. Now you can hear salesforce.com podcasts—covering everything from luminary interviews and thought-leadership presentations to roundtable discussions and best practices—whenever, wherever you like. Choose from over 70 podcasts. Happy listening!

Figure 103: Salesforce.com on iTunes

AppExchange Partner Program

With the emergence of The Business Web™, companies can offer their services and applications to businesses over the Internet as easily as retailers and auctioneers can connect with online consumers. As AppExchange partners, more than 150 companies are already participating in this new chapter of computing by making their offerings available on The Business Web via the AppExchange. The AppExchange partner program makes it easy for both new and established businesses to join this growing community of on-demand providers.

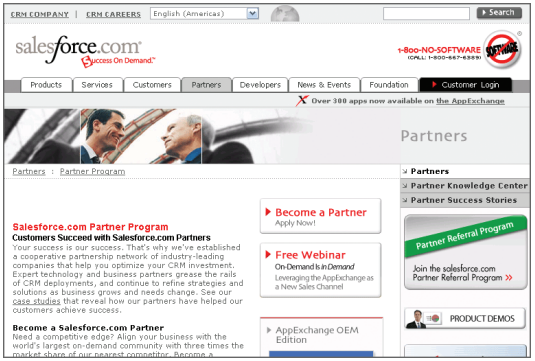


Figure 104: AppExchange Partner Website

Visit the Partner Program website at www.salesforce.com/partners/ and join the AppExchange partner program today!

What Do You Think?

Well that about sums it up. Did you like what you've read? Has it inspired you to go out and create your own on-demand apps? We certainly hope so! We welcome any comments you might have—indeed, we count on your feedback and ideas. Go to the Apex Developer Network discussion boards at community.salesforce.com/sforce?category.id=developers or email us at adn@salesforce.com and let us know what you think!

Glossary

Apex Developer Network (ADN)
The website at www.salesforce.com/developer/ that provides a full range of resources for Force.com developers, including sample code, toolkits, an online developer community, and the test environments necessary for building apps.

App
A collection of Force.com components such as tabs, reports, dashboards, and custom s-controls that address a specific business need. Short for "application."

AppExchange app menu
A menu that enables users to switch between customizable applications (or "apps") with a single click. The AppExchange app menu displays at the top of every page in the Salesforce user interface.

AppExchange directory
A Web directory where hundreds of AppExchange apps are available to Salesforce customers to review, demo, comment upon, and/or install. Developers can submit their apps for listing on AppExchange if they wish to share them with the community.

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 process
An automated process your organization can use to approve records on the platform. An approval process specifies the steps necessary for a record to be approved and who must approve it at each step. Approval processes also specify the actions to take when a record is approved, rejected, or first submitted for approval.

Auto number
A custom field type that automatically adds a unique sequential number to each record.

Client app
An app that runs outside the Salesforce user interface and uses only the API—typically running 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* on page 250 and *Native app* on page 252.

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* on page 250 and *Native app* on page 252.

Controlling field
Any standard or custom picklist or checkbox field whose values control the available values in one or more corresponding dependent fields. See also *Dependent field* on page 250.

Custom field
Fields that can be added to customize an object for your organization's needs.

Custom link
A custom URL defined by an administrator to integrate your data with external websites and back-office systems.

Custom object
An entity that you build to store information that's unique to your app. See also *Object* on page 252 and *Standard object* on page 255.

Customer Relationship Management (CRM)
A business strategy for developing and improving relations between companies and their customers.

Dashboard
A graphical representation of data from up to 20 summary or matrix reports arranged in a two- or three-column layout. Every user can select a favorite dashboard to display on his or her Home tab.

Database
An organized collection of information. The underlying architecture of Force.com 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* on page 252.

Dependent field
Any custom picklist or multi-select picklist field that displays available values based on the value selected in its corresponding controlling field. See also *Controlling field* on page 250.

Developer Edition account
A free, fully-functional salesforce.com account that allows you to get hands-on experience with all aspects of the platform in an environment designed for development. Developer Edition accounts are available on the Apex Developer Network website at www.salesforce.com/developer/.

Email template
A built-in feature that enables you to create form emails that communicate a standard message, such as a welcome letter to new employees or an acknowledgement that a customer service request has been received.

Entity relationship diagram (ERD)
A data modeling tool that helps you organize your data into entities (or objects, as they are called in Force.com terms) and define the relationships between them.

Field
A part of an object that holds a specific piece of information, such as a text or currency value.

Field dependency
A filter that allows you to change the contents of a picklist based on the value of another field.

Field-level security
Settings that determine whether fields are hidden, visible, read only, or editable for users based on their profiles.

Force.com
A platform for building on-demand applications from salesforce.com. Force.com combines a powerful user interface, operating system, and database to allow you to customize and deploy on-demand applications for your entire enterprise.

Force.com Web Services API
An application programming interface that defines a Web service that provides direct access to all data stored on the platform from virtually any programming language and platform. See also *Application programming interface (API)* on page 249.

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. See also *Primary key* on page 253.

Formula field
A type of custom field that automatically calculate its value 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.

Group
A set of users that can contain individual users, other groups, or the users in a role. Groups can be used to help define sharing access to data.

Home tab
The starting page from which users can view a dashboard, choose sidebar shortcuts and options, view current tasks and activities, or select each of the major tabs.

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. See also *S-Control* on page 255.

Junction object
A custom object that enables a many-to-many relationship between two other objects.

Layout
See *Page layout* on page 253.

Lookup relationship
A relationship between two objects that allows you to associate records with each other. On one side of the relationship, a lookup field allows users to click a lookup icon and select another record from a list. On the associated record, you can then display a related list to show all of the records that have been linked to it.

Manual sharing
Record-level access rule that allows record owners to give read and edit permissions to other users who might not have access to the record any other way. See also *Record-level security* on page 254.

Matrix report
A report that allows you to group and subtotal records by two unrelated values—one in the rows dimension and one in the columns dimension. Matrix reports are the most complex and powerful reports on the platform.

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-driven development
An app development model that allows apps to be defined as declarative "blueprints," with no code required. AppExchange apps—their data models, objects, forms, workflows, and more—are defined by metadata.

Multitenancy
An application model where all users and apps share a single, common infrastructure and code base.

Native app
A type of AppExchange app that is built exclusively via metadata configuration and without coding. Native apps run entirely on the platform without need for external services or infrastructure. See also *Client app* on page 250 and *Composite app* on page 250

Object
In Force.com terms, an object is similar to a database table--a list of information, presented with rows and columns, about the person, thing, or concept you want

to track. Each object automatically has built-in features like a user interface, a security and sharing model, workflow processes, and much more.

Object-level security
Settings that allow an administrator to hide whole tabs and objects from a user, so that they don't even know that type of data exists. On the platform, you set object-level access rules with object permissions on user profiles. See also *Field-level security* on page 251 and *Record-level security* on page 254.

On-demand computing
A new paradigm of computing in which you access apps over the network as a utility, rather than as pieces of software running on your desktop or in the server room.

One-to-many relationship
A relationship in which a single object is related to many other objects. For example, each Candidate may have one or more related Job Applications.

Org-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's enabled in their user profile, but that they'll need extra permissions to actually edit one.

Page layout
The organization of fields, custom links, related lists, and other components on a record detail or edit page. Use page layouts primarily for organizing pages for your users, rather than for security.

Picklist
A selection list of options available for specific fields, for example, the `Country` field for a Candidate object. Users can choose a single value from a list of options rather than make an entry directly in the field.

Picklist values
The selections displayed in drop-down lists for particular fields. Some values come predefined, and other values can be changed or defined by an administrator.

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. See also *Foreign key* on page 251.

Profile
A component of the platform that defines a user's permission to perform different functions. The platform includes a set of standard profiles with every organization, and administrators can also define custom profiles to satisfy business needs.

Queue
A collection of records that don't have an owner. Users who have access to a queue can examine every record that's in it and claim ownership of the records they want.

Record
A single instance of an object. For example, Software Engineer is a single Position object record.

Record-level security
A method of controlling data in which we can allow particular users to view and edit an object, but then restrict the individual object records that they're allowed to see. See also *Org-wide defaults* on page 253, *Role hierarchy* on page 254, *Sharing rules* on page 254, and *Manual sharing* on page 252.

Related list
A section of a record or other detail page that lists items related to that record.

Relationship
A connection between two objects in which matching values in a specified field in both objects are used to link related data. For example, if one object stores data about companies and another object stores data about people, a relationship allows you to find out which people work at the company.

Report
A summary of stored data, including a table and an optional graph, filters, and groupings. See also *Tabular report* on page 255, *Summary report* on page 255, and *Matrix report* on page 252

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. See also *Record-level security* on page 254.

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.

Search layout
The organization of fields included in search results, lookup dialogs, and the recent items lists on tab home pages.

Sharing model
A security model that defines the default organization-wide access levels that users have to each other's information.

Sharing rules
Rules that 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. Sharing rules also allow administrators to make automatic exceptions to org-wide defaults for particular groups of users.

Snippet
A type of s-control that's 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. See also *S-Control* on page 255.

SOAP (Simple Object Access Protocol)
A protocol that defines a uniform way of passing XML-encoded data.

SOQL (Salesforce 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 the data from the platform database.

Standard object
A built-in object included with the platform. You can also build custom objects to store information that's unique to your app. See also *Custom object* on page 250 and *Object* on page 252.

S-Control
A component that you set up and define on the platform to store custom code. Use s-controls to create and display custom data forms using components like custom links, web tabs, or custom buttons. For example, you can define a custom s-control containing JavaScript and address merge fields to display a map of a contact's address. See also *HTML s-control* on page 252, *URL s-control* on page 255, and *Snippet* on page 254.

Summary report
A report that's similar to a tabular report, except that it also allows users to group and subtotal rows of data, and create graphs. See also *Tabular report* on page 255.

Tab
An interface item 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.

Tabular report
Similar to a spreadsheet, a report that includes an ordered set of fields as columns and a matching record in each row. Tabular reports are best for creating lists of records or a list with a single grand total.

Time-dependent workflow action
A workflow action that occurs before or after a certain amount of time has elapsed. Time-dependent workflow actions can fire tasks, field updates, outbound messages, and email alerts while the condition of a workflow rule remains true.

Time trigger
A setting that defines when time-dependent workflow actions should fire.

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 ultimately hosted by an external website. See also *S-Control* on page 255.

Validation rule
A rule that prevents a record from being saved if it does not meet the standards that are specified.

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.

Web tab
A custom tab that allows your users to use external websites from within the application.

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 on-demand 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 "container" for a set of workflow instructions that includes the criteria for when the workflow should be activated, as well as the particular tasks, alerts, and field updates that should take place when the criteria for that rule are met.

Workflow task
A workflow action that assigns a task to an application user when a workflow rule is triggered.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y