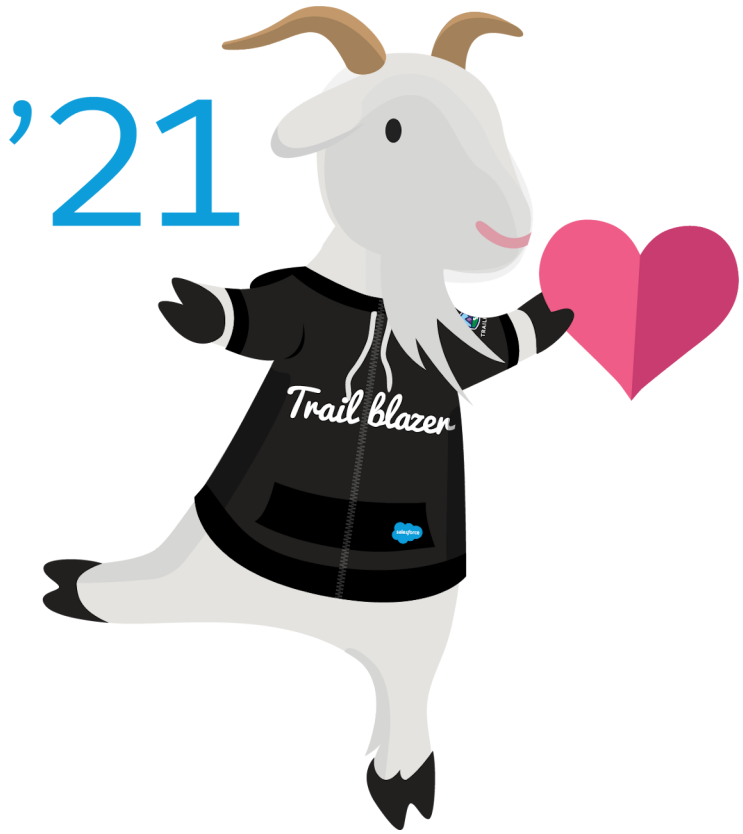
Chatter Administrator's Workbook

Version 2, Summer '21



@salesforcedocs

Last updated: May 18, 2021

© Copyright 2000–2021 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

ABOUT THE CHATTER WORKBOOK

Chatter adds a rich suite of collaboration features to any application built on Lightning Platform, enabling creative solutions for your business needs.

The Chatter Administrator's Workbook is an introduction to the major components of Chatter, as well as to administering Chatter in your organization. We'll even show you how to develop a few of your own applications using Chatter features.

Supported Browsers

For information about supported browsers, see Supported Browsers for Salesforce Classic in the Salesforce help.

Before You Begin

You will need a Lightning Platform environment that supports both Chatter and Lightning Platform development. These tutorials are designed to work with a Lightning Platform Developer Edition environment, which you can get for free at https://developer.salesforce.com/signup.

How is the Workbook Organized?

The five tutorials can be completed in any order, though we recommend completing Tutorial 1 and Tutorial 2 first.

- Tutorial 1 explores the fundamentals. It walks you through using the standard Chatter application, which introduces you to many of the basic underlying concepts, such as posts, feeds, groups, and comments.
- Tutorial 2 shows you how to better increase collaboration in your organization by adding Chatter Free users.
- Tutorial 3 walks you through building a very simple Chatter application on Lightning Platform. You'll learn how to interact with the data model that supports Chatter, which is the starting point of any application built with Chatter.
- Tutorial 4 walks you through adding a trigger that checks for a specific phrase, then acting on that phrase. This example lets the owner of an opportunity close it by posting `!close` to the opportunity feed.
- Tutorial 5 shows you two different ways to remove feed items in bulk.

Tell Me More....

At the end of each step, there is an optional Tell Me More section. If you like to do things quickly, move on to the next step. However, if you're a smell-the-roses type, there's a lot of useful information here.

- To continue exploring Chatter development, check out the Chatter Cheat Sheet at `https://developer.salesforce.com/page/Cheat_Sheets`.
- To learn more about Lightning Platform and to access a rich set of resources, visit Salesforce Developers at `https://developer.salesforce.com`.
- For a gentle introduction to developing on Lightning Platform, see the companion Lightning Platform Workbook at `https://developer.salesforce.com/page/Force.com_workbook`.

- For more developer resources related to Chatter, visit the Chatter Resource Page on Salesforce Developers: https://developer.salesforce.com/page/An_Introduction_to_Salesforce_Chatter.

TUTORIAL #1: ORIENTATION AND SETUP

Level: Beginner; **Duration:** 25 minutes

Chatter provides a rich suite of features, including user profiles, feed updates, comments, groups, and feeds. You can use these features to add a collaborative and social dimension to your Lightning Platform applications. Chatter also lets data records play a part in this collaboration—so people not only follow other people, but also data that's important to them. In effect, your applications can bring people and data closer to each other.

In this tutorial you will explore the basic Chatter functionality. Along the way you'll learn about users, posts, status updates, comments, feeds, following users and data, and groups. You'll also learn the names of the API objects that model these features in the Chatter data model. All of these concepts are used in later tutorials.

Prerequisite

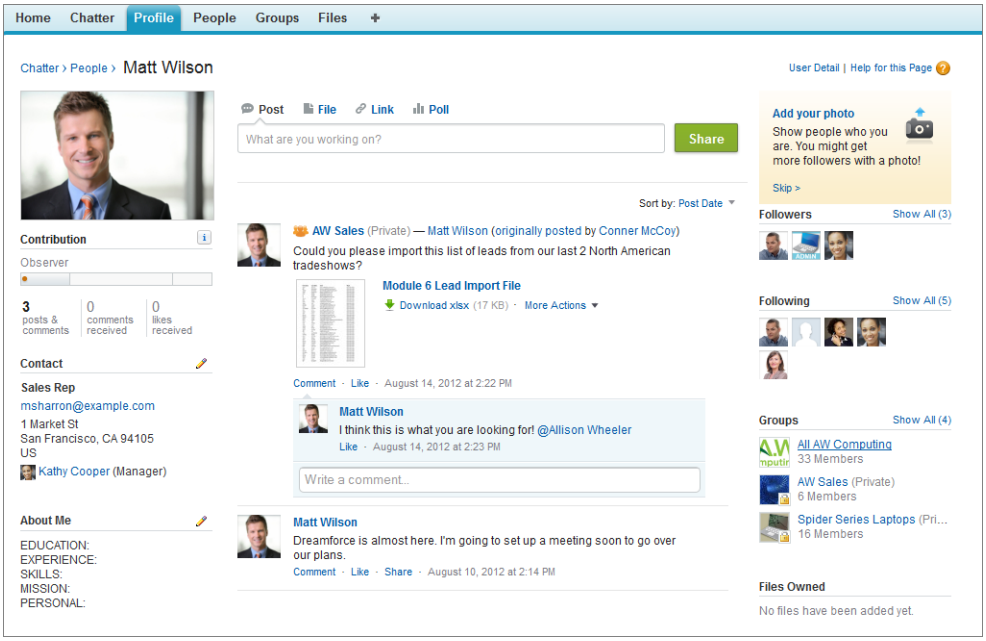
Chatter

This workbook requires an environment with Chatter enabled. We recommend using a free Developer Edition environment. To sign up, go to `https://developer.salesforce.com/signup`.

Step 1: Set Up Your Chatter Profile

In any collaboration environment, it's important to configure your profile so that other users can easily identify and find you. The first step is to log in, find the Chatter application, update your profile, and create your first status update.

1. In your browser, go to `https://login.salesforce.com`.
2. Enter your username (in the form of an email address) and password.
3. In the top right-hand corner, select **Salesforce Chatter** from the drop-down list of applications.
4. Select the **Profile** tab. It displays your posts and record updates, and posts that other people have made on your profile.



5. Click the pencil icon in the **About Me** area and add your role and interests to the description. Other people can use this description to determine if they want to follow you. Click **Save**.
6. Optionally, select **Add Photo** and upload a photograph of yourself.
7. Finally, make a new update. Click in the text box and replace **What are you working on?** with some text, such as *`My first Chatter post.`* Click **Share**.

Congratulations, you have just set up your user profile and made your first post. In Chatter, you can follow data, as well as other users. In the next step you'll follow some data, and in a later step you'll create a user to follow.

Tell Me More....

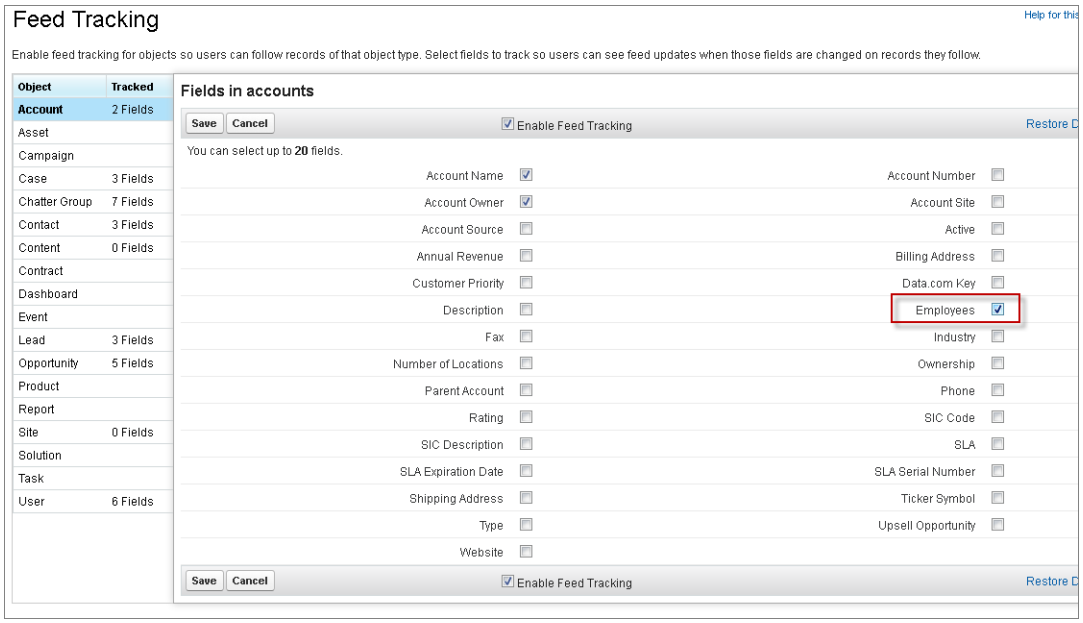
- The **Profile** tab also lists the users that follow you, as well as users and data records that you are following.
- When you post on someone else's profile, the `UserProfileFeed` record has `TextPost` as the value in the `Type` field, unless you also post a file or a link. If your post contains a file, it is a `ContentPost`, or if it contains a link, a `LinkPost`. You can also leave a comment on a post, which is stored in the `FeedComment` object.
- You can also use the Recommendations area in the upper right-hand corner of the Chatter or Profile tabs to get more information or perform tasks such as adding your photo or creating a group.

Step 2: Follow Data

In this step, you're going to follow a data record to see how changes to the record are reflected in your Chatter feed. Before you can follow records, you need to enable feed tracking.

1. From Setup, enter *`Feed Tracking`* in the `Quick Find` box, then select **Feed Tracking**.
2. Select the **Account** object. Selecting an object displays its tracked fields.

3. Enable the checkbox in the **Employees** field and click **Save**.



To see the effects in action, you need to follow an account record and modify the Employees field.

1. In the top right-hand corner, select the **Sales** application from the drop-down list.
2. Click the **Accounts** tab.
3. Select **All Accounts** from the view filter, then click **Follow** for any account.

4. When you make a change to this account, anyone who follows the account sees the post in their Chatter feed.
5. Now click **Edit** to edit the account.
6. Navigate to the `Employees` field, and change the number to *200*. Click **Save**.

When you modify a tracked field, all followers see the change you made. To see this, click the Chatter tab. This tab displays the news feed—a list of all posts and updates by users and records you follow, or groups of which you are a member. You'll see a new update, this time from the account that you followed, indicating that a change was made to the `Employees` field.

Chatter lets you follow data records, as well as have conversations about them. Each data record in the default user interface now displays the related Chatter posts (called a record feed), while also providing an indication of who else is following the data.

Tell Me More....

- You can follow both standard objects, such as Accounts, as well as any custom object you create, in exactly the same manner.
- When one of the fields you're following is changed in a record, the update appears on your user profile on the Profile tab. The update is also tracked on the `AccountFeed` record. All objects have feeds, for example, `ContactFeed`, `LeadFeed`, or even `MyCustomObject__Feed`.

- Chatter always indicates the users and records you follow on your Profile tab. Because you now follow a record, the record's name is included in your list of people and records you're following.
- Chatter posts can be made by people or by records. Chatter posts by records have the value `TrackedChanges` in the `Type` field of the record that represents the post. Together with `TextPosts`, `ContentPosts`, and `LinkPosts`, these cover some of the types of posts that users can make in Chatter.

Step 3: Follow Users

Chatter is a collaborative platform, so you'll need more than one user if you want to collaborate. In this step, you create a new user, log in as that user, follow the original user and comment on their status update. You need to have your email address handy as user creation involves receiving an email with new user details.

1. From Setup, enter *`Users`* in the `Quick Find` box, then select **Users**, and then click **New User**.
2. Enter the following data in the fields provided:
   - First Name: *`Joe`*
   - Last Name: *`Blogs`*
   - Alias: *`jblogs`*
   - Email: enter a valid email address you can check
   - Username: create a unique email address, such as adding *`chatter-`* in front of your email address

     Note: A username must be unique and in the form of an email address. You can have multiple usernames that use the same address. For example, you could have usernames of `chatter-mary@mydomain.com` and `admin-mary@anotherdomain.com`, that both have the same actual email address.

   - User License: if an option is presented, select **Salesforce** (not Salesforce Platform)
   - Profile: select **System Administrator**
3. Click **Save** and then click *`Your Name`* > **Logout**.

You'll receive an email indicating the username you selected, and a temporary password. Log in as Joe Blogs:

1. Navigate to `https://login.salesforce.com/`
2. Enter Joe Blogs' username and temporary password.
3. At the prompt, enter a new password and security questions.

Now you're ready to work as Joe Blogs.

1. Select the **Salesforce Chatter** application.
2. Click the **People** tab.
3. Find your profile (the first one you created) and click **Follow**.
4. Click on your name. The list of posts is the `UserProfileFeed`. Find the post you made, click the **Comment** link and type some text. Click **Comment** to save.
5. Click *`Your Name`* > **Logout**.

Now log back in as yourself and follow Joe:

1. Navigate to `https://login.salesforce.com/` and log in with your credentials.
2. Select the Chatter app.
3. Click the **Profile** tab. The page now indicates that Joe Blogs is following you, and you can see the comment he left.

4. Select Joe Blogs, by clicking on his name, or image. Click **Follow**. Now you're following Joe as well.

Tell Me More....

- You can follow a user by looking them up in the People tab, or by clicking **Follow** when viewing their profile.
- You'll see an email from Chatter informing you that Joe Blogs left a comment. To toggle email notifications, from Setup, enter *`Chatter Setting`* in the `Quick Find` box, select **Chatter Settings**, and then modify the Email Notification Settings.

Step 4: Create a Chatter Group

In a collaboration environment, people often form groups to share knowledge or accomplish a task. Groups are also a good way to filter the information being generated by your users.

In this step, you create and join a Chatter group.

1. Click the **Groups** tab.
2. Click **New Group**.
3. Enter a name for the group, and optionally, a description and click **Save**.
4. Click **Add Members** and add Joe Blogs to the group.
5. Now post an update to the group. In the field that displays "Share with *`Group Name`*" type *`My first group update!`* and click **Share**. Everyone in the group (you and Joe Blogs) sees that update.

Tell Me More....

- Updates to any group are represented by a `CollaborationGroupFeed` record.
- Each group member sees an update from the group in their Chatter feed, which is a news feed.
- Groups can be public, private, or hidden. In public groups, anyone can view the updates, post to the group, and join. In private groups, you must explicitly approve members before they can join. In hidden groups, members must be explicitly asked to join before they can see the group.

Summary

In this tutorial, you learned how to create a Chatter profile, follow users, leave comments, and create groups. This basic Chatter functionality introduced you to some of the different types of posts that you can create in Chatter.

While you worked through this first tutorial, you might have noticed a number of different views on all the posts—we call these views *feeds*. The feed on your Profile tab lists posts or tracked changes that you've made, or that target you. The feed on group's page lists updates that are made to the group. Each record of an object also displays a feed of posts to that record, or tracked changes made to the record, called a record feed. You started following a record feed, which in this case, was an account feed. Finally, the feed on the Chatter tab displays your updates, updates of people you follow, updates to records you follow, and groups that you are a member of.

TUTORIAL #2: INCREASE COMMUNICATION IN YOUR ORGANIZATION USING CHATTER FREE

Level: Beginner; **Duration:** 15 minutes

Suppose that your management wants to increase communication in your company, but they don't want to increase the budget. One of the ways to address this is to add Chatter Free users to your organization. Every organization has 5,000 Chatter Free licenses. These users can access standard Chatter people, profiles, groups, and files. They can't access any Salesforce objects or data.

There are two ways of adding Chatter Free users:

- By an administrator adding Chatter Free users.
- By invites, which allows current Chatter users to invite fellow employees to join Chatter.

In this tutorial, you'll practice adding users using both methods. In addition, you'll learn more about the functionality of Chatter, such as using @mentions and # topic tags.

Step 1: Adding Chatter Free Users

In this tutorial, you'll add users to Chatter Free.

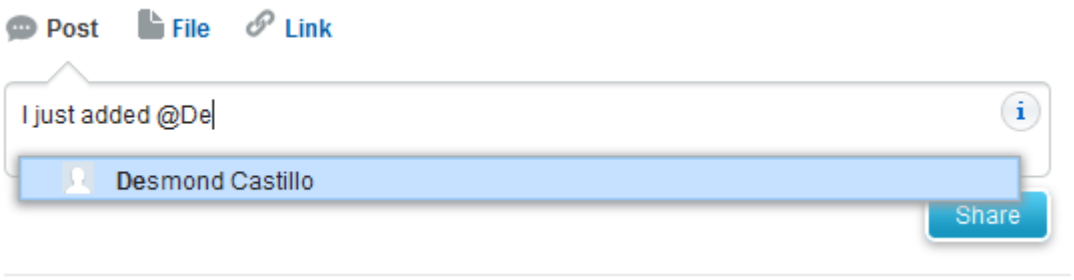
1. From Setup, enter *Users* in the Quick Find box, then select **Users** then click **Add Multiple Users**.

   Note: You can also create just a single user by clicking **New User**.

2. Select the **Chatter Free** user license.
3. Add users, such as the following:
   - First name/last name: Desmond Castillo
   - Email (user name): DCastillo@*yourcompany.com*
   - Profile: Chatter Free user
   - First name/last name: Francine Dubois
   - Email (user name): FDubois@*yourcompany.com*
   - Profile: Chatter Free user
   - First name/last name: Gabriel Jones
   - Email (user name): GJones@*yourcompany.com*
   - Profile: Chatter Free Moderator user
4. When you click **Save**, all the users are created.

Now, you can mention those users in a new post, follow them, add them to groups, and so on.

1. Click the Chatter tab.
2. Post to your group, "I just added @des". Note how when you start typing after the @ sign, you are prompted to select from people you follow, as well as other people who are in your organization:



3. Continue with the message, ending with a topic tag, such as #NewMembers. Topic tags are a way of tagging your posts so you can search them easily later, as well as grouping them.

Tell Me More....

- If you look at the profile of the user you granted Chatter Free Moderator, you'll see their user picture is slight different. Moderators have special administration privileges in Chatter, such as deleting inappropriate posts or deactivating users that left the company.
- On your Chatter page there's a section called **Trending Topics** that you can use for following all topics. You can even save your topic tag as a search to use again later.

Step 2: Enable Chatter Invites

In the last step, you added individual users to Chatter, using Chatter Free licenses.

In this step, you enable the users in your organization to invite their co-workers to Chatter. Again, people who join using these invitations are Chatter Free users, and only have access to Chatter, not to Salesforce, such as accounts, cases, and so on.

1. From Setup, enter "Chatter Settings" in the `Quick Find` box, then select **Chatter Settings** then click **Edit**.
2. Click **Allow Invitations**.
3. Enter email domains to define who can join Chatter for your organization. The domains you enter should include those used in email addresses for your company, such as *`yourcompany.com`*.
4. Click **Save**.
5. Click the Chatter tab. Note that now there's a button for inviting coworkers to Chatter.

Tell Me More....

- Only people with email from the domain you specified are able to join.
- Groups can replace email group lists.
- Groups can unite users with shared interests in products, company initiatives and more.

Summary

In this tutorial you learned how to add Chatter Free users to your organization. You also learned more about using Chatter, including functionality such as mentions and topic hash tags.

TUTORIAL #3: BUILDING A CHATTER APP ON THE PLATFORM

Level: Intermediate; **Duration:** 30 minutes

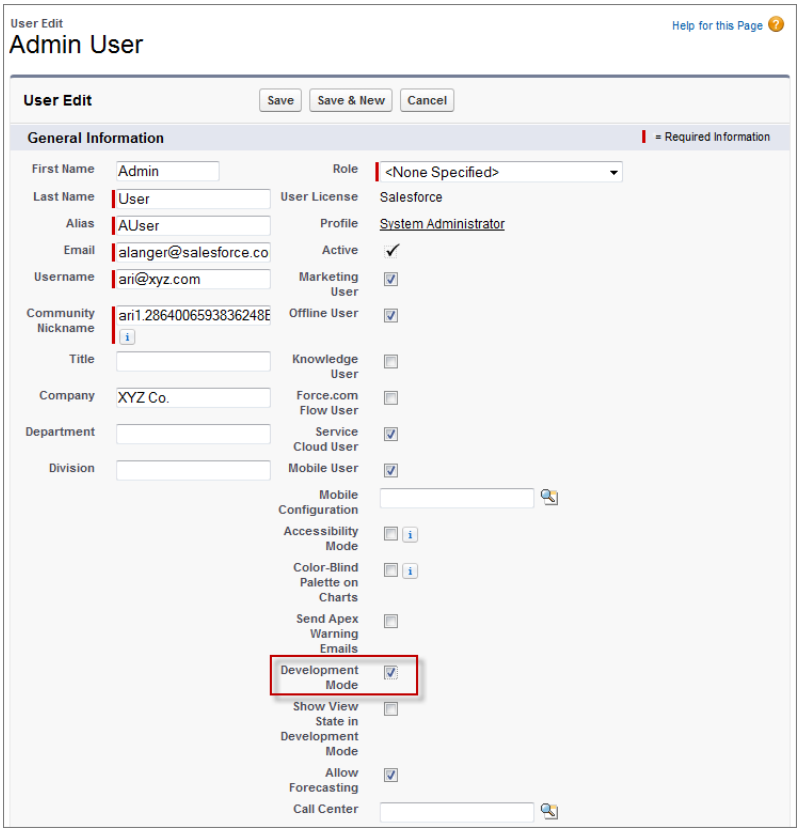
Chatter is built on the Lightning Platform platform. As a result, you can extend the Chatter functionality, or enhance your own applications with Chatter features. In this tutorial, you take some of the feed concepts from the first tutorial and build a Visualforce page that lets you programmatically update a feed and display it.

Step 1: Create a Visualforce Page

In this step, you use Visualforce to create a custom user interface. Visualforce is a framework that allows developers to create custom pages that are hosted natively on the Lightning Platform platform.

You also create a page to list your news feed and display an input box for updating your feed. Before you create the page, you should enable Development Mode, which embeds a page editor into Visualforce pages, simplifying development.

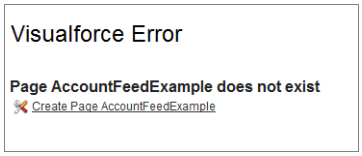
1. From your personal settings, enter `Advanced User Details` in the Quick Find box, then select **Advanced User Details**. No results? Enter `Personal Information` in the Quick Find box, then select **Personal Information**.
2. Select the **Development Mode** checkbox and click **Save**.



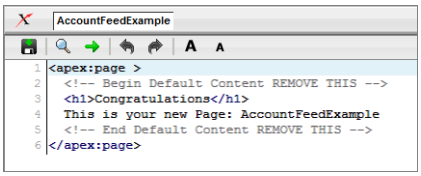
3. In your browser, add the text `/apex/AccountFeedExample` to the current URL. For example, if your current URL is `https://MyDomainName.my.salesforce.com/005A0000000hFv5`, the new URL is `https://MyDomainName.my.salesforce.com/apex/AccountFeedExample`. You receive an error message: "Page AccountFeedExample does not exist."

   Note: Unsupported browsers do not show this error message. To create the page, from Setup, enter *Visualforce Pages* in the Quick Find box, then select **Visualforce Pages**. Next, click **New**. After you create the page, go back to the URL.

4. Click the **Create Page AccountFeedExample** link to create the new page.



5. Click **AccountFeedExample** in the bottom left corner of the browser. The Page Editor tab displays the code and a preview of the new page (which has some default text).



6. You don't really want the heading of the page to say "Congratulations," so change the contents of the heading tag to `Account Feed` and remove both the comments and the contents of the page. While you're there, add an attribute `sidebar="false"` to the `apex:page` tag. Your code should be as follows:

```
<apex:page sidebar="false">
  <h1>Account Feed</h1>
</apex:page>
```

7. When you click the save icon , notice that the page has dynamically updated.

Tell Me More....

- Notice that the code for the page looks like standard HTML. That's because a Visualforce-specific tags.
- The built-in page editor includes tag completion, as well as a comprehensive component reference that documents how each of the Visualforce components work. Click `Component Reference` to access the reference documentation.

Step 2: Add User Update Functionality

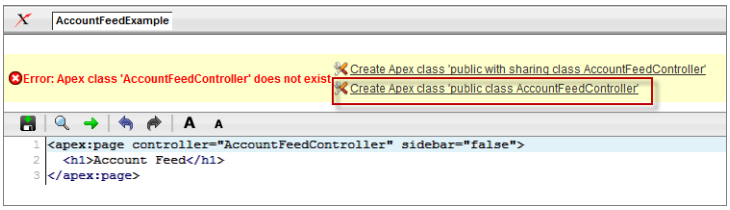
Lightning Platform uses the Model-View-Controller (MVC) design pattern to separate the view and its styling from the underlying database and logic. In MVC, the view (the Visualforce page) interacts with a controller. In our case, the controller is an Apex class, which exposes some functionality to the page. For example, the controller contains the logic that executes when clicking a button. A controller also typically interacts with the model (the database, that is, your information in Salesforce)—exposing data that the view displays.

In this step, you modify the Visualforce page to include a text area and button, and add logic to update your Chatter feed with the value of the text area after a user clicks the button.

1. If the Page Editor isn't open on your Visualforce page, click **AccountFeedExample** in the bottom left corner.
2. Modify your code to include a `AccountFeedController` custom controller by editing the `<apex:page>` tag.

```
<apex:page controller="AccountFeedController" sidebar="false">
```

3. When you click the save icon, the editor notes that the `AccountFeedController` class doesn't exist, and offers to create it for you. Click **Create Apex class 'public class AccountFeedController'**.



> **Note:** If this message doesn't appear, you may be using an unsupported browser. To create the class, from Setup, enter *Apex Classes* in the Quick Find box, then select **Apex Classes** , and click **New**. In the class, add the text `AccountFeedController { }`, and then return to the URL of your Visualforce page.

4. Just after the closing `</h1>` tag, add a form, together with an input text area and a button:

```
<apex:form>
     <apex:inputText value="{!feedUpdate}"/>
     <apex:commandButton value="Update" action="{!go}" reRender="recent"/>
</apex:form>
```

5. Click the save icon. The editor informs you that you don't have a property called 'AccountFeedController.feedUpdate' and offers to create it for you. Select **Create Apex property 'AccountFeedController.feedUpdate'**.
6. The editor informs you that you don't have a method called `go`. Click **Create Apex method 'AccountFeedController.go'**.

By now you might have noticed that the page has been updated with a text entry box, as well as an update button. However, this functionality isn't working yet, you've only updated the page.

Next to **AccountFeedExample**, an **AccountFeedController** tab now displays. The editor has created some of the code for you. When you click **AccountFeedController**, you see the generated code below:

```
public class AccountFeedController {

   public PageReference go() {
      return null;
   }
   public String feedUpdate { get; set; }
}
```

When a value is entered into the text area, the `feedUpdate` property that you just created is assigned the value. When you click the **Update** button, the Visualforce page calls the `go()` method on the controller. You want this method to update the user's news feed with the value of the field. That's the next and final task.

1. Modify the contents of the `go()` method to query the User object and locate the currently logged-in user with `UserInfo.getUserId()`.

```
public PageReference go() {
  User u = [SELECT id FROM User WHERE id = :UserInfo.getUserId()];
  return null;
}
```

2. Now create the feed item, with the value found in the `feedUpdate` property, and then update the `FeedItem` record in the database.

```
public PageReference go() {
  User u = [SELECT id FROM User WHERE id = :UserInfo.getUserId()];
        FeedItem fItem = new FeedItem();
        fItem.Type = 'TextPost';
        fItem.ParentId = u.id;
        fItem.Body = this.feedUpdate;
        insert fItem;
  return null;
}
```

3. Be sure to click the save icon.

Tell Me More....

- The Visualforce `PageReference` type, used in the `go()` method, determines the page that displays after the method completes. By returning `null`, you indicate that no new page displays.
- When you want to update someone's Chatter feed, you create a new feed item.

Step 3: Display and Query a Feed

The Chatter feed on the Home and the Chatter tab displays your posts and posts by people and objects you follow. Let's display only text posts after each update, ignoring comments, posts with attachments, link posts, and so on.

1. Add a new `getRecentTextUpdates()` method to the `AccountFeedController` that you created in the previous steps. After the first line in the controller, add the following:

```
public List<AccountFeed> getRecentTextUpdates() {
}
```

2. Now use a query to retrieve the list of `AccountFeed` items that are `Text` updates. You'll want to order the posts by date, the most recent being first:

```
public List<AccountFeed> getRecentTextUpdates() {
   List <AccountFeed> aAccountFeed =
        [SELECT  Type, CreatedDate, CreatedBy.name, Body
         FROM AccountFeed WHERE Type='TextPost'
         ORDER BY CreatedDate DESC LIMIT 10];
   return aAccountFeed;
}
```

3. The code now returns a list of `AccountFeed` items that are `TextPost` updates. So far so good, but now modify the Visualforce page to display them. Click **AccountFeedExample** to show the panel displaying your Visualforce markup.
4. Underneath the line containing `<apex:commandButton>`, insert the following:

```
<apex:outputPanel id="recent">

</apex:outputPanel>
```

5. You might have noticed that the command button you added earlier has an attribute `reRender="recent"`. This ensures that an AJAX page update is made when the button is clicked—refreshing a block on the page identified by the `"recent"` identifier. Go ahead and put your list within that block. Enter the following between the `outputPanel` tags:

```
<apex:dataList value="{!recentTextUpdates}" var="update">
   <apex:outputText value="{!update.body}"/>
</apex:dataList>
```

6. Click **Save**.
7. Enter some text and click the **Update** button. You'll see the page dynamically list the most recent text posts beneath the query box.

Tell Me More....

- The `<apex:dataList>` component iterates over a list, assigning each item in the list to a variable. In our sample code, it assigns each item (an instance of `AccountFeed`) to a variable named `update`. The `<apex:outputText>` component outputs values like the body of the `AccountFeed` record. For text post updates, the `body` field holds the text that was posted as part of the update.
- The Visualforce page doesn't output all the values that are retrieved from the query. For example, `CreatedDate` and `CreatedBy.Name` are ignored. As an optional exercise, extend the Visualforce page to display these fields as well.

Step 4: Add an Automated Test

In this step, you add tests to Apex to test your controller. Testing ensures that your code behaves as it should and doesn't consume resources unnecessarily. Not only does testing benefit you (you will have more confidence in the applications you write), but it also benefits the platform. The platform requires that you test any Apex you write before that code can be deployed to a production environment. Tests are also run before a new release of the platform is made available, to ensure that your code doesn't introduce backward compatibility problems.

Apex supports unit tests to facilitate testing, which programmatically validate the code behavior and expected results. All Apex code has a set of limits that determine how many resources can be consumed. For example, there is a limit on the number of queries that can be performed in a trigger. Good tests ensure that your code not only behaves as it should, but also that it doesn't exceed these limits. In this step you write and execute a unit test, which exercises the Apex controller you just created.

1. From Setup, enter "Apex Classes" in the `Quick Find` box, then select **Apex Classes**.
2. Click on the name of your class, `AccountFeedController` and click **Edit**.
3. Create a new method before the final closing brace:

```
@isTest
static void testUpdates() {
}
```

4. Between the curly braces, instantiate a new instance of your controller class:

```
AccountFeedController h = new AccountFeedController();
```

5. Assign the `feedUpdate` property and then call the `go()` method to post the update.

```
h.feedUpdate = 'hi';
h.go();
```

6. Return the list of status updates.

```
List<AccountFeed> after = h.getRecentTextUpdates();
```

7. Now you need to test that the retrieved list has at least one element, and that the most recent update corresponds to the status update you just made. You know the most recent post is first because of the `ORDER BY` in the original query.

```
System.assert(after.size() > 0 );
System.assertEquals('hi', after[0].body);
```

8. Verify your finished code looks like the following and then click **Quick Save**.

```
@isTest
static void testUpdates() {
   AccountFeedController h = new AccountFeedController();
   h.feedUpdate = 'hi';
   h.go();
   List<AccountFeed> after = h.getRecentStatusUpdates();
   System.assert(after.size() > 0 );
   System.assertEquals('hi', after[0].body);
}
```

   If there are any errors in your code, for example, if you misspelled a variable or forgot a period, you have the opportunity to correct those errors now.

9. Click **Save**.

10. Click **Run All Tests**. If all goes well, your screen indicates that you ran one test, and that its code coverage is 100%.

Tell Me More....

- Testing is a critical part of code development. This was a very simple example. In a real-world scenario, your tests verify both positive and negative results for many different use cases.
- The `@isTest` annotation indicates that the method is a test method, and that it must be called during testing.

Summary

The data model behind Chatter can be accessed from an application on Lightning Platform, letting you create applications that manipulate or query the data related to the Chatter functionality. In this tutorial you created logic that updated a user's status and queried their news feed. You can imagine embedding some of this logic in your own applications, or extending it using the examples from the Chatter Cheat Sheet at `https://developer.salesforce.com/page/Cheat_Sheets`.

Recommended Apps

You don't necessarily have to create your own applications for Chatter. Much of the expanded functionality has already been developed for you and is available on the AppExchange. Here's a list of some of the more popular and useful applications:

Name and Link	Description
Chatter Usage Dashboards	The Chatter Adoption Dashboard includes 20 dashboard components and reports for a broad view into your organization's usage of Chatter. Extend it with your own customized reports by using the 7 included Chatter report types!
Chatter Combo Pack	A package of a number of different apps: • 7 tabbed Chatter apps • 5 Chatter apps for your sidebar • 5 Visualforce components that you can use to build your own Chatter apps
Email2Chatter	Email2Chatter lets you forward emails directly into Chatter. Every user has their own private and secure email that they can send updates to. They can even tag emails with #account #opportunity #contact or #case to post updates directly to the record. Email2Chatter even takes attachments and posts them to the feed.
Cloud Swarm	Cloud Swarm allows users to automatically follow leads, opportunities and cases in Chatter based on the rule criteria you define. Or set up your own custom swarms that are triggered by Chatter posts. (There is both a managed and an unmanaged version of this app.)
Unfollow Rules	When you or your users create new records, they automatically follow the records. However, many records have a limited useful Chatter life such as when a case or opportunity closes. Also, there's currently a 500 following limit for each user. The Chatter Unfollow Rules app makes it easy to create rules that save everyone in your organization from those issues! Create rules to remove followers from records that match your criteria. (There is both a managed and an unmanaged version of this app.)

TUTORIAL #4: USING CHATTER OBJECTS IN AN APEX TRIGGER

Level: Advanced; **Duration:** 30 minutes

Lightning Platform exposes all of the Chatter functionality via the API objects. To review:

User Interface	Commonly Called	API object
Chatter tab	News feed	News feed resource in the Connect REST API
Profile tab	User profile feed	User-profile feed resource in the Connect REST API
Groups tab	Group feed	`CollaborationGroupFeed`
Feed displayed on an object tab, such as Account, Contact, Lead, and so on	Record feed	Depends on the object. Examples include: `AccountFeed`, `ContactFeed`, `MyCustomObject__Feed`

You can use Apex triggers to monitor for posts for specific keywords. Suppose you have agents in the field who want to be able to close opportunities using the Salesforce mobile app. You could monitor all posts on opportunity records, looking for a particular keyword, then close the opportunity programmatically.

In this tutorial, you use Apex to monitor opportunity posts to close opportunities when a user posts `!close`, as well as assign tasks based on the closed opportunity.

You can do similar things monitoring other record types, such as cases.

Step 1: Create an Apex Trigger Definition

In this step, you create the trigger definition, which contains the trigger name, the affected object, and an action that fires the trigger.

1. In Salesforce, from Setup, enter *`FeedItem`* in the `Quick Find` box, then select **FeedItem Triggers**.
2. Click **New**.
3. Replace `<name>` and `<events>` so that the code matches the following:

```
trigger CloseOpportunity on FeedItem (after insert) {

}
```

4. Click **Quick Save**, which saves your work and lets you continue editing. Saving your work at this point also verifies that you've entered the code correctly, because if you've made syntax errors, the system won't let you save.

Tell Me More....

Your trigger doesn't do anything yet, but it's now ready to accept any logic that you want to execute when a feed item is added. Before we get to the trigger logic, let's break up the trigger definition and examine each part.

- `CloseOpportunity`—The name of the trigger.

- `FeedItem`—The object the triggers acts on. Remember you can't place triggers on record feed objects, such as `OpportunityFeed` or `MyCustomObject__Feed`. You can only place triggers on the `FeedItem` and `FeedTrackedChanged` objects.
- `after insert`—The action that fires the trigger. Apex triggers are fired in response to an action, such as a record insert, update or delete, either before or after one of these events. The trigger you defined fires after the record (feed item) is inserted.
- `{ }`—The code that goes between the curly brackets is called the *body* and determines what the trigger does. You'll code some of the trigger body next.

Step 2: Limit the Trigger to Opportunities

The first thing you need to do is to make sure that you're limiting your actions to just opportunity records. You can use the same mechanism to limit your actions to just accounts, users, custom objects, or any other type of record.

1. Between the curly braces of your trigger definition, add the following variables:

```
   Set<Id> oppIds = new Set<Id>();
   List<Task> tasks = new List<Task>();
   List<Opportunity> opps2Update = new List<Opportunity>();
   Map<Id, Id> oppId2PostCreator = new Map<Id, Id>();
```

2. After the variables, enter the following comment, then the describe call:

```
// Get the key prefix for the Opportunity object
// using a describe call.
String oppKeyPrefix = Opportunity.sObjectType.getDescribe().
   getKeyPrefix();
```

3. Click **Quick Save** to make sure the variables and call are entered correctly.
4. Add the following logic to limit the trigger to operate just on opportunity records that have a feed item that contains `!close` as the start of the body of the post.

```
for (FeedItem f: trigger.new)
{
   String parentId = f.parentId;
   // Compare the start of the parentID field
   // to the Opportunity key prefix. Use this to
   // restrict the trigger to act only on posts made to the
   // Opportunity object.
   if (parentId.startsWith(oppKeyPrefix) &&
       f.type == 'TextPost' &&
      f.Body.startsWith('!close'))
   {
       oppIds.add(f.parentId);
       oppId2PostCreator.put(f.parentId, f.CreatedById);
   }
}
```

5. Click **Quick Save** to save your work and verify your syntax.

Tell Me More....

Let's dig a little deeper into the code:

- `getKeyPrefix`—Returns the three-character prefix code for the object, in this case, an opportunity. Record IDs are prefixed with three-character codes that specify the type of the object (for example, accounts have a prefix of `001` and opportunities have a prefix of `006`).
- `for (FeedItem f: trigger.new)`—Iterates through all new items in the trigger, that is, everything that's been added to the `FeedItem` record.
- The following code checks to see if the record is an opportunity, if the post is a text post, and if the body starts with the key phrase `!close`.

```
   if (parentId.startsWith(oppKeyPrefix) &&
       f.type == 'TextPost' &&
       f.Body.startsWith('!close'))
```

Step 3: Add Business Logic

Now you're going to add the business logic to close the opportunity and add a new task associated with the closed opportunity.

1. Add the following to select the appropriate fields for the opportunities:

```
List<Opportunity> opps = [SELECT id, account.name, ownerId,
   stageName, closeDate
   FROM Opportunity
   WHERE id IN :oppIds];
```

2. Add the following to close the opportunity and add a task:

```
 for (Opportunity o : opps)
   {
   //Compare the creator of the Chatter post
   //to the opportunity owner to ensure
   //that only authorized users can close the opportunity
   //using the special code !close
    if (oppId2PostCreator.get(o.Id) == o.ownerId)
    {
        o.StageName = 'Closed Won';
        o.CloseDate = System.today();

       Task t = new Task ( OwnerId = o.OwnerId,
                           WhatId = o.Id,
                           Priority = 'High',
                           Description = 'Check-in with '+account.name,
                           Subject = 'Follow-up',
                           ActivityDate = System.today().addDays(7));
          tasks.add(t);
          opps2Update.add(o);
      }
  }
```

3. Update the opportunity and add the tasks:

```
    update opps2Update;
    insert tasks;
```

4. Click **Save**. Your entire code should look like this:

```
trigger CloseOpportunity on FeedItem (after insert) {
     Set<Id> oppIds = new Set<Id>();
     List<Task> tasks = new List<Task>();
     List<Opportunity> opps2Update = new List<Opportunity>();
     Map<Id, Id> oppId2PostCreator = new Map<Id, Id>();

    // Get the key prefix for the Opportunity object
    // using a describe call.
     String oppKeyPrefix = Opportunity.sObjectType.getDescribe().
       getKeyPrefix();

     for (FeedItem f : trigger.new)
     {
         String parentId = f.parentId;
         //Compare the start of the parentID field to the
         //Opportunity key prefix to restrict the trigger to
         //act on posts made to the Opportunity object.
         if (parentId.startsWith(oppKeyPrefix) &&
             f.type == 'TextPost' &&
             f.Body.startsWith('!close'))
         {
             oppIds.add(f.parentId);
             oppId2PostCreator.put(f.parentId, f.CreatedById);
         }
     }

     List<Opportunity> opps = [SELECT id, account.name, ownerId,
                               stageName, closeDate
                               FROM Opportunity
                               WHERE id IN :oppIds];

     for (Opportunity o : opps)
     {
        //Compare the creator of the Chatter post to the
        //Opportunity Owner to ensure
        //that only authorized users can close the Opportunity
        //using the !close
         if (oppId2PostCreator.get(o.Id) == o.ownerId)
         {
             o.StageName = 'Closed Won';
             o.CloseDate = System.today();

             Task t = new Task ( OwnerId = o.OwnerId,
                                 WhatId = o.Id,
                                 Priority = 'High',
                                 Description = 'Check-in with '+account.name,
                                 Subject = 'Follow-up',
                                 ActivityDate = System.today().addDays(7));
```

```
            tasks.add(t);
            opps2Update.add(o);
        }
    }

    update opps2Update;
    insert tasks;
}
```

Tell Me More....

The final statement, `update opps2Update`, updates the opportunity records in the database. That's simple enough, but what about that `for` loop?

- `for (Opportunity o : opps)`—Iterates over the list of opportunities.
- `if (oppId2PostCreator.get(o.Id) == o.ownerId)`—Verifies that the owner of opportunity is who posted the `!close` message.
- `Task t = new Task . . .`—Creates a task associated with the opportunity closing, then inserts all tasks.

Step 4: Test the Trigger in Salesforce

In this step, test to make sure your code actually closes an opportunity.

1. Create a new opportunity. Click the Opportunity tab, then click **New**.
2. Enter a name for the opportunity, such as MyTestOpp. Select a close date in the future, and Stage Negotiation/Review, then click **Save**.
3. If you are not automatically following the opportunity record, click **Follow**.
4. Now post the keyword `!close` to the opportunity and click **Share**.
5. Refresh the page. You will see the opportunity has been closed, the close date changed to the current date. If you check your tasks, you'll see a new one has been added. (You may need to look at All Open tasks.)

Step 5: Add Tests

While you already know the trigger works, automated tests are required before you can deploy the code to a production environment. In this step, you add a single test method to ensure the trigger is working as expected, that is, that it closes the opportunity as well as creates a task.

1. Create the test class. From Setup, enter "Apex Classes" in the `Quick Find` box, then select **Apex Classes**, then click **New**.
2. In the editor pane, enter the following code:

   ```
   @isTest
   private class TestCloseOppTrigger {
   }
   ```

   The `@isTest` annotation tells Lightning Platform that all code within the class is code that tests the rest of your code.
3. Click **Quick Save** to save your work and continue editing.

4. Add the test method to do the actual testing. Between the curly braces for `TestCloseOppTrigger`, add the following code:

```
    static testMethod void testCloseOpportunity() {

    }
```

5. Between the curly braces for `testCloseOpportunity()`, add the following code, which creates an account and an opportunity for testing:

```
        // Create a test account
        Account testAcct = new Account (Name = 'My Test Account');
        insert testAcct;

        // Create a test opportunity
        Opportunity testOpp = new Opportunity( Name = 'My Test Opp',
                                    AccountId = testAcct.Id,
                                    StageName = 'Qualification',
                                    CloseDate = System.today().addDays(4));
        insert  testOpp;
```

6. Directly below the `insert testOpp;`, add the following code which creates a post on the opportunity created above, that contains the phrase `!close`. This code executes the trigger:

```
        // Create a feed post on the test opportunity
        FeedItem fpost = new FeedItem();
        fpost.ParentId = testOpp.Id;
        fpost.body = '!close';
        fpost.type = 'TextPost';
        insert fpost;
```

7. Now write the code that verifies that the trigger returns the expected results:

```
        // Get the opportunity, verify the stage name and close date
        testOpp = [SELECT StageName, CloseDate
                   FROM Opportunity
                   WHERE id = :testOpp.Id];

        System.assertEquals(testOpp.StageName, 'Closed Won');
        System.assertEquals(testOpp.CloseDate, System.today());

        //  Get the task that was supposed to be created based on
        // closing the opportunity
        List<Task> tsks = [SELECT id
                           FROM Task
                           WHERE WhatId = :testOpp.Id];

        System.assertEquals(tsks.size(), 1);
```

8. Verify that your code looks like the following and then click **Save**:

```
@isTest
private class TestCloseOppTrigger {

    static testMethod void testCloseOpportunity() {
```

```
        // Create a test account
        Account testAcct = new Account (Name = 'My Test Account');
        insert testAcct;

        // Create a test opportunity
        Opportunity testOpp = new Opportunity( Name = 'My Test Opp',
                                  AccountId = testAcct.Id,
                                  StageName = 'Qualification',
                                  CloseDate = System.today().addDays(4));
          insert  testOpp;

        // Create a feed post on the test opportunity
        FeedItem fpost = new FeedItem();
        fpost.ParentId = testOpp.Id;
        fpost.body = '!close';
        fpost.type = 'TextPost';
        insert fpost;


        // Get the opportunity, verify the stage name and close date
        testOpp = [SELECT StageName, CloseDate
                   FROM Opportunity
                   WHERE id = :testOpp.Id];

        System.assertEquals(testOpp.StageName, 'Closed Won');
        System.assertEquals(testOpp.CloseDate, System.today());

        //  Get the task that was supposed to be created based on
        // closing the opportunity
        List<Task> tsks = [SELECT id
                           FROM Task
                           WHERE WhatId = :testOpp.Id];

        System.assertEquals(tsks.size(), 1);
    }
}
```

Tell Me More....

- Creating your own test data is a standard part of writing any tests, as well as asserting the results at the end.
- The `@isTest` annotation indicates that the method is a test method, and that it must be called during testing.
- Testing is a critical part of code development. This was a very simple example and only tested a single record. Your tests should verify a bulk situation as well.

Step 6: Execute the Test

Lightning Platform has a testing framework that lets you execute tests, and also check code coverage. You are now going to execute the test and look at the resulting code coverage.

1. From Setup, enter "Apex Test Execution" in the `Quick Find` box, then select **Apex Test Execution**, then click **Select Tests...**
2. Select the `TestCloseOppTrigger` class then click **Run**. Note that you could select all the tests in your organization.

3. When the test has finished running, click the name of the test to populate the view pane (below), then click **View** to view the test results.

Tell Me More....

- To calculate the code coverage for your entire organization, from Setup, enter *Apex Classes* in the Quick Find box, then select **Apex Classes**. Then select **Estimate your organization's code coverage**. This number is based on the last test runs in your organization.
- You can also run a test by viewing the test details and then clicking **Run Test**.

Summary

In this tutorial, you wrote an Apex trigger that looks for opportunity records that have a specific keyword, then closes that opportunity, as well as assigns a task. You also created a test to ensure that this code works correctly, then executed that test using the Apex Test Execution page.

You can explore the other methods on objects to learn what other actions you can perform, or consult the Chatter cheat sheet to find queries to run against the Chatter data model.

TUTORIAL #5: DELETING CHATTER DATA

Level: Advanced; **Duration:** 30 minutes

You can specify the individual fields that generate a Chatter post for the records you are following. Suppose that you specified that you wanted to follow all account address changes. However, that evening you've scheduled a batch processing of all accounts, and over a million records get changed.

Your news feed is now full of information you don't need or want. In addition, you could now be approaching the storage limits for your organization.

How do you remove these feed items? In this tutorial, we show you several different ways to manage your feed.

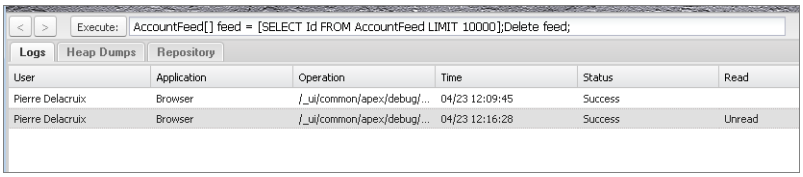
Warning: When you delete feed items, they are not moved to the Recycle Bin. They are hard deleted, and you cannot recover them.

Step 1: Delete Feeds Using the Developer Console

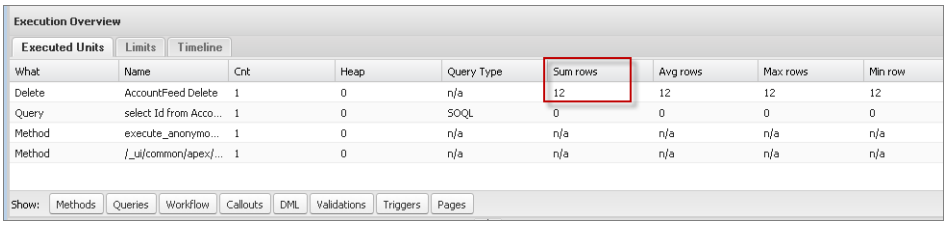
1. Open the Developer Console under `Your Name` or the quick access menu ( ).
2. Click **Debug** > **Open Execute Anonymous Window**.
3. In the popup window, enter the following, then click **Execute**.

```
AccountFeed[] feed = [SELECT Id FROM AccountFeed LIMIT 10000];
Delete feed;
```

4. Once the code finishes running, double click the log to display it in the workspace. If you have more than one log, it's generally the one that's unread.



5. At the bottom of the Developer Console, in the Execution Overview section, look at the `Delete` method on the **Executed Units** tab to see how many records were deleted:



Tell Me More....

You can only process up to 10,000 records at a time using Apex. This limit is called a governor limit. For more information on Apex governor limits, see the *Apex Developer Guide*.

The Developer Console is an integrated development environment with a collection of tools you can use to create, debug, and test applications in your Salesforce org.

Step 2: Delete Feeds Using Batch Apex

In the previous step, you used the Developer Console to delete less than 10,000 records. Suppose you needed to delete more than 10,000 records?

You could write a simple program using batch Apex to process up to 50 million records.

1. From Setup, enter "Apex Classes" in the `Quick Find` box, then select **Apex Classes**, then click **New**.
2. In the text box, enter the following code:

```
// To use batch Apex, you must implement the Database.Batchable interface
global class DeleteChatterFeed implements Database.Batchable<sObject>{

   // Add a comment to the debug log
   // This helps you keep track when you have many records to process
   global DeleteChatterFeed() {
     System.Debug('Deleting Feed');
   }

   // The Database.Batchable interface has three methods
   // you must define: start, execute, and finish
   // Start defines the size of the job, and takes one million records
   global Database.QueryLocator start(Database.BatchableContext BC) {
     return Database.getQueryLocator('SELECT Id FROM AccountFeed LIMIT 1000000');
   }

   // Execute defines size of each batch of records
   global void execute(Database.BatchableContext BC, List<sObject> scope) {
     delete scope;
   }

   // Finish could be used to send yourself email once the job is over
   global void finish(Database.BatchableContext BC) {
   }

}
```

3. Click **Save**.
4. To execute this class, open the Developer Console under `Your Name` or the quick access menu ( ).
5. Click **Debug** > **Open Execute Anonymous Window**.
6. Enter the following and then click **Execute**.

```
DFC batchinstanceid = database.executeBatch(new DeleteChatterFeed(), 10000);
```

This sends the batch job to the Apex job queue.

7. To look at the progress of the job, from Setup, enter "Apex Jobs" in the `Quick Find` box, then select **Apex Jobs**.

Tell Me More....

The `10000` in the code to execute the class is the size of each batch job. It is passed to the `execute` method as the `scope`. You can experiment with this number to see what works best for your batch runs. Jobs with a smaller scope tend to be processed more quickly.

You can use batch Apex to run complex, long-running jobs as well. For more information on batch Apex, see the *Apex Developer Guide*.

Summary

In this tutorial, you used the Developer Console to delete a smaller number of items from a feed. Then you used batch Apex to delete a larger number of items from a feed.


Warning: When you delete feed items, they are not moved to the Recycle Bin. They are hard deleted, and you cannot recover them.

If you need to delete an even larger number of items, you can use either Data Loader or Bulk API. For more information, see the Data Loader Guide and the Bulk API 2.0 and Bulk API Developer Guide.