Complete Guide to Social Customer Service

Salesforce, Spring '17

@salesforcedocs

Last updated: March 11, 2017

CONTENTS

IMPLEMENTING SOCIAL CUSTOMER SERVICE

Welcome to Social Customer Service

Social Customer Service lets you create cases or leads out of social media posts and send personalized responses on the same social media channels.

Note: The Social Customer Service setup pages, the moderation and authorization pages, and the Reply action in the case feed are only available in Salesforce Classic.

Important: If you have two or fewer social accounts to track, you can use the free starter pack. Otherwise, you must have sufficient Social Studio accounts.

Watch the following video for a brief overview on getting started with Social Customer Service: Support Your Customers on Their Social Networks (Salesforce Classic)(2:58).

Social Customer Service integrates with Radian6 and Social Studio so service agents and sales representatives can engage customers by responding to cases and leads created from Facebook, Twitter, Instagram, and other social networks.

Social Network	Release State
Facebook	Generally Available
Twitter	Generally Available
Instagram	Generally Available
Google+	Pilot Program
Sina Weibo	Pilot Program

The social publisher action on the case or lead feed is the primary interface for replying to consumers or prospects. Inbound and outbound social posts appear as items in the feed, making it easy to follow the thread of the conversation. Permission sets allow you to grant access to your managed social accounts to different sets of users. Out of the box, default settings control how inbound social posts are processed. Optionally, you can modify an Apex class to apply your own custom business logic.

Note: When a lead is converted to an account or contact, the social items in the feed are removed.

In Salesforce1, agents can see and reply to social content from mobile devices.

For Twitter accounts, agents can use case and lead feeds to see the content that they are responding to, retweet, mark as Like and follow tweets, send replies to tweets and direct messages, and delete tweets managed by your social accounts.

For Facebook accounts, cases and leads are created from your managed Facebook page and agents can use the feeds to see the content that they are replying to, like posts and comments, send posts, comments, replies, and private messages, and delete posts managed by your social accounts.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To administer Social Customer Service:

- "Manage Users"

  AND

  "Customize Application"

To create case feed items:

- Feed Tracking for All Related Objects on the Case object

To send and receive social media posts or messages:

- Case Feed enabled

  AND

  Access to a social account

Implementing Social Customer Service

For both Facebook and Twitter, you can click the View Source link to open an item in its native social media Web site. This allows you to gain context so you can provide a more informed response.

You can sync your Instagram #hashtag to Social Customer Service to receive Social posts when your brand's #hashtag is mentioned on Instagram. To activate Instagram #hashtag listening, create a rule in your Social Hub account to receive posts when your brand's #hashtag is mentioned.

Social Customer Service Limits	**Enterprise, Performance, and Unlimited editions.**
Maximum number of active managed social accounts	2000 accounts. Social Customer Service Settings shows up to 500 managed social accounts per page.
Maximum number of post tags	200 post tags
Maximum number of errors before inbound posts are paused	100 errors. When errors reach 100, inbound posts are stopped until the errors are cleared. This allows administrators to ensure that posts are processed correctly.

SEE ALSO:

- Implementing Social Customer Service
- Social Action Tips

Implementing Social Customer Service

Enable social customer service in your organization and customize your support agents' experience.

- Set Up Social Customer Service
- Create the Social Action Interface
- Enable Moderation for Social Customer Service
- Modify the Default Apex Class

SEE ALSO:

- Social Action Tips
- Manage Social Posts
- Manage Social Personas

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To administer Social Customer Service:

- "Manage Users"

  AND

  "Customize Application"

To create case feed items:

- Feed Tracking for All Related Objects on the Case object

Set Up Social Customer Service

Enable Social Customer Service, install the SocialCustomerService package, sync your social accounts, and assign social handles.

Important: If you have two or fewer social accounts to track, you can use the free starter pack. Otherwise, you must have sufficient Social Studio accounts.

Radian6, part of Marketing Cloud's Social Studio, provides social media monitoring. With the free starter pack, you don't need an explicit Radian6 login. Simply connect up to two social accounts and Salesforce handles the rest of the details. If you are a Social Studio customer, set up data sources in Social Studio that can be synced with Service Cloud.

Important: Case Feed Tracking for All Related Objects must be enabled for case feed items to be created. See Set Up Cases for Salesforce Classic. For Leads, from Setup, enter *Feed Tracking* in the `Quick Find` box, then select **Feed Tracking** and ensure `Enable Feed Tracking` and `All Related Objects` are checked. When a lead is converted to an account or contact, the social items in the feed are removed.

1. From Setup, enter *Social Media* in the `Quick Find` box, then select **Social Customer Service**.

   Tip: Enter *med* as a shortcut to bring up the Social Media section more quickly.

2. On the Settings tab, check `Enable Social Customer Service`.
3. If you want posts approved before they send, check `Enable approvals for social posts`.

   As part of a job training or quality review process, you may require some agents to have their posts approved rather than allowing them to post freely. With approval processes and user permissions, selected agents can submit social posts for approval, recall the posts, and retry or resubmit them. Approvers can approve and reject posts for publication. See Enable Social Post Approvals on page 7.

4. If you want to map new posts to parent posts, which are the first posts that generated a case, select **Enable retrieval of parent posts for added context**.
5. Under `Radian6 Credentials`, either create a Social Studio account with the starter pack by clicking **Create Account**, or click **Login** and enter your Social Studio credentials.

   Note: With the Social Customer Service Starter Pack, you can enable Social Customer Service and up to two social accounts from any social network. For example, if you add one Twitter account, you can only add one Facebook account. You can't downgrade from a Social Studio account to the starter pack. The starter pack doesn't support the moderation feature (all posts become cases) and the default Apex code can't be customized.

6. On the Social Accounts tab, click **Add Account** and select your social network, for example Twitter or Facebook.

   The social network opens and asks you to authenticate the account. Once your account is authenticated, Salesforce returns you to the Social Accounts tab.

   Note: If you receive an error "We're sorry, but we currently do not support Facebook business accounts registration." or "Your Facebook account can't be added due to unsupported features.", you might need to set a user name on your Facebook page.

7. Click the refresh icon next to **Add Account**.

   Warning: Deleting a Social Account deletes it everywhere, including in Social Studio. This action cannot be undone.

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To administer Social Customer Service:

- "Manage Users"

  AND

  "Customize Application"

To create case feed items:

- Feed Tracking for All Related Objects on the Case object

8. If you are using the Starter Pack, check the `Case Creation` box to indicate that you want cases created automatically when posts come from the social account.

   For example, if you have two Twitter handles, one for support and one for marketing or brand-focused information, you can have cases created automatically only from the support handle. The tweets from the marketing handle go in a social post queue for review. See Manage Social Posts on page 69.

   Note: If you are using the full Social Customer Service version, you can set up case moderation through Social Studio. See Enable Moderation for Social Customer Service on page 8.

9. If you have a portfolio of managed social accounts, set the `Default Responses From` for each Twitter, Instagram (pilot), and Sina Weibo (pilot) account. This lets you standardize and raise awareness of your brand's support by setting a dedicated support handle, for example @acmehelp or @acmesupport. Also, agents have fewer clicks when they send outbound posts because the chosen account appears as the default value in the account drop-down in the social publisher. The default response handle doesn't apply for Twitter direct messages and doesn't affect Facebook, Google Plus, or LinkedIn, as they are restricted to the page handle itself.

10. On the `Inbound Settings` tab, you can see which Apex class controls how the inbound content is processed in your organization and which user it's set to run under. If you are using the default Apex class, you can select inbound business rules to determine how incoming social data is handled.

    Enable Case Reopen
    If a new post, from the same social persona, is associated to a closed case, the case is reopened within the designated number of days. The number must be greater than or equal to 1 and less than or equal to 3000.

    Use Person Accounts
    Assign a person account of the selected type for the social persona parent record.

    Create Case for Post Tags
    Override the social hub's case creation rules and create a case when selected post tags are present on a social post. Post Tags are used to answer the question "What is the topic of this one post?". Post tags, set in Social Hub, help to provide further context to what the individual post is about.

    Note: Social Customer Service only shows 200 post tags. If you have more post tags in Social Studio, you can view them in that program.

    The default Apex class creates a social post, social persona, case, contact, and supports common use cases. For information on modifying the default Apex class, see Modify the Default Apex Class.

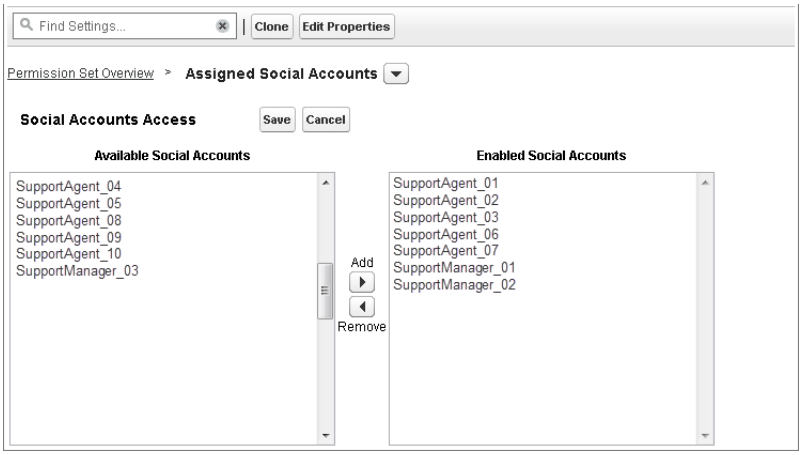
    Note: If you are using the starter pack, the Apex class can't be changed but you can change the user it is run under.

11. To assign social handles to a profile or permission set, still within Setup:
    - Enter *`Profiles`* in the `Quick Find` box, then select **Profiles**.
    - Enter *`Permission Sets`* in the `Quick Find` box, then select **Permission Sets**.

12. Click an existing profile or permission set or create a new one.

13. In the Apps section, click **Assigned Social Accounts**.

14. Click **Edit**.



15. Assign the social accounts you need to make available to your users with this profile or permission set.

    Important: All users must have the profile or permission set you chose or created in step 8.

16. Save your changes.
17. Ensure that the profile or permission set has the correct field visibility.
    - For profiles, from Setup, enter `Profiles` in the `Quick Find` box, select **Profiles**, then select the profile you chose or created earlier. Next, in the Field Level Security section, select **Social Post**.
    - For permission sets, from Setup, enter `Permission Sets` in the `Quick Find` box, select **Permission Sets**, then select the permission set you chose or created earlier. Next, click **Object Settings**, and then select **Social Post**.
18. Click **Edit**. Under Field Permissions, ensure all fields available are set to Visible (not Read-Only) for profiles or Edit for permission sets. Click **Save**.
19. Optionally, set up Quick Text so agents can create ready-to-send responses to social networks. See Enable Quick Text.
20. Optionally, give social post read access to external community and portal users.

    There are three requirements to make social posts available in communities and portals.
    - Ensure that the user has access to cases in the community.
    - Give users read permission to social posts on their profiles.
    - On your organization's Social Post object, enable visibility to individual fields through the field level security settings.

    Note: Once these requirements are met, external users can see all social posts exposed to them. For example, if a case or lead feed is exposed externally, all social posts in the feed are visible. There is no way to limit visibility at the social post object level.

Turning on history tracking on for the Social Persona and Social Post objects is recommended for the first few months of using Social Customer Service. History tracking helps identify who made what changes when and for differentiating between automatic and manual changes.

You can synchronize up to 2,000 managed social accounts from Social Studio. However, the Social Customer Service Settings page in Setup only shows up to 500 managed social accounts. Agents can respond from all synced accounts from the social publisher on the case feed. If you are syncing more than 500 social accounts, allow at least a minute for the settings page to load.

If you use Process Builder to automate business processes note that it isn't currently supported for the social post and persona objects. You'll need to use a workflow.

SEE ALSO:

Salesforce Help: Field History Tracking

How to Reconnect a Social Account

Reconnect your social account for Social Customer Service.

Your social account can be disconnected from Social Customer Service if your social media network provider's connect, or token, has expired. Many providers have expiration policies of 60 to 90 days.

1. From Setup, enter *Social Media* in the Quick Find box, then select **Settings**.
2. On the Social Accounts tab, click **Reauthorize** in the Action column.

   The social network opens and asks you to authenticate the account. Once your account is reauthenticated, you are returned to the Social Accounts tab.

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To administer Social Customer Service:

- "Manage Users"

  AND

  "Customize Application"

To create case feed items:

- Feed Tracking for All Related Objects on the Case object

To send and receive social media posts or messages:

- Case Feed enabled

  AND

  Access to a social account

Enable Social Post Approvals

Social care agents are both problem solvers for your consumers and the voice of your brand on social networks like Facebook and Twitter. You can have guidelines so your agents write with a consistent tone and syntax that's in line with your organization's social media strategy. For example, you require social agents to sign their tweets in a standard manner, such as "~John."

Salesforce Admins can create approval processes and assign agents and approvers permissions accordingly.

1. From Setup, enter *Social Media* in the Quick Find box, then select **Settings**.
2. Select Enable approvals for social posts.
3. Build and activate approval processes for social posts using either the Jump Start Wizard or the Standard Setup Wizard.

   Important: The Jump Start Wizard is a streamlined way to create approval processes in Salesforce. However, the Let the submitter choose the approver manually option is not supported in the Jump Start Wizard. Choosing that option results in an error later when an agent submits a post for approval.

4. From Setup, go to **Administer** > **Manage Users** > **Permission sets**.
5. Enable the new Require Social Post Approvals user permission.
6. Assign the Require Social Post Approvals user permission with a permission set to agents that need their posts reviewed before they are sent.

   When assigning user permissions, remember these two points.

   - Because approving a post automatically submits it for publishing, approvers must have the same access to social accounts as the agents whose work they're reviewing. Otherwise, the posts they approve result in an error.
   - If your user permissions include Require Social Post Approvals, then the submit button on the social publisher always reads **Submit for Approval** rather than "Comment," "Tweet," or other words. This is true even if no active approval process applies to the user. In that situation, clicking **Submit for Approval** publishes the social post normally since there is no active approval process in effect.

For more information, see Create an Approval Process with the Standard Wizard, Prepare to Create an Approval Process, and Sample Approval Processes.

Tip: If your agents work with social post record detail pages, rather than in the case feed, we recommend removing the approvals related list from the page layout. The same page layout is shared between inbound and outbound social posts. Removing the approvals related list avoids confusion when viewing an inbound post that is an invalid candidate for an approval process. Approvers can still approve or reject posts through all other normal means such as email, Chatter, and list views.

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To administer Social Customer Service:

- "Manage Users"

  AND

  "Customize Application"

To create case feed items:

- Feed Tracking for All Related Objects on the Case object

Enable Moderation for Social Customer Service

Use moderation to triage incoming posts and only create cases for posts that are actionable requests for help. Moderation helps your organization focus on real customer issues and avoid opening unnecessary cases.

Not all posts require a case, for example, a complimentary tweet or post does not need agent assistance. However, when the default social customer service is configured, cases are automatically created from each social post. Using moderation, agents can manage which posts get cases and which are ignored. Moderation is enabled with a Social Hub rule in your Social Studio account to turn off automatic case creation.

Note: With the Starter Pack, you can decide if you want cases created automatically when posts come from a particular social account on the Social Accounts tab. See Set Up Social Customer Service on page 4.

1. From your Social Hub account, click the **Rules** tab.
2. Create a rule, or use an existing one, to indicate that no case is created in Salesforce.

   For example, the rule should have the following setup.

   a. Action: send to Salesforce.

   b. `Create Case` checkbox unchecked.
3. Save and enable your rule.

Note: You can enable your rule for all social posts or only those coming from certain managed accounts.

Case creation can also be customized by implementing a custom Apex case logic. To do so, from setup, enter *`Social Media`* in the `Quick Find` box, then select **Settings**. See Modify the Default Apex Class.

Note: If you started using Social Customer Service before Spring '16 and have a custom Apex class, you may need update your Apex class to benefit from the latest moderation features. If your custom Apex is extended from the default Apex class, you get the update for the default apex functions you call. If your custom Apex isn't extended from the default Apex class (you copied the default and changed it), you must update manually.

To manually update your custom Apex class, add the following code and update your moderation social post list view.

1. Call this method directly before inserting the post, after all the relationships have been set on the post.

```
private void setModeration(SocialPost post){
    //if we don't automatically create a case, we should flag the post as requiring
 moderator review.
    if(post.parentId == null)
        post.reviewedStatus = 'Needed';
}
```

   In the default Apex, see lines 50 and 61-65.

2. Update your moderation social post list filter from:

```
Parent EQUAL TO "" AND ReviewStatus NOT EQUAL TO "ignore"
```

   To:

```
Parent EQUAL TO "" AND ReviewStatus EQUAL TO "Needed"
```

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To administer Social Customer Service:

- "Manage Users"

  AND

  "Customize Application"

To ensure that you don't lose track of social posts currently in your moderation queue, make a list view with the new filter, and switch to it once the new and old filters show the same results.

Create the Social Action Interface

The social action is created when you install Social Customer Service. You can add, remove, and organize fields to suit your organization.

The social action is created when Social Customer Service is enabled.

1. From the object management settings for cases, go to Button, Links, and Actions.
2. Click **Layout** next to the social action.
3. Edit the desired fields.

   Note: Changing field values could invalidate incoming posts against the Social Customer Service Apex class.

   To send social content, the social action must have the following fields:

   - `In Reply To`

     Important: The In Reply To field can't be read only.

   - `Managed Social Account`
   - `Message Type`
   - `Content`

   Headline and Name are required fields. To remove them, create a predefined value for each field and remove them from the action. See Set Predefined Field Values for Quick Action Fields.
4. Click **Save**.
5. From the object management settings for cases, go to Page Layouts.
6. In Case Page Layouts, click **Edit** next to Feed-Based Layout.
7. In the palette, click **Quick Actions**.
8. Ensure that the social action is in the Quick Actions in the Salesforce Classic Publisher section of the layout.
9. Optionally, repeat steps 5 through 8 for the Leads object to enable the social action on leads (from the object management settings for leads, go to Page Layouts).

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To administer Social Customer Service:

- "Manage Users"

  AND

  "Customize Application"

To create case feed items:

- Feed Tracking for All Related Objects on the Case object

Modify the Default Apex Class

If you aren't using the Starter Pack, you can customize the default Apex class to specify how inbound social content is processed.

Note: You can't modify the default Apex class if you are using the Starter Pack. The free Starter Pack lets you simply connect up to two social accounts and Salesforce handles the rest of the details, like a Social Studio account.

The default Apex class for Social Customer Service creates a social post, social persona, case, contact, and supports common use cases. To customize how information is processed, by create a new Apex class.

Important: If your agents use the Social Customer Service feature to send private messages to Facebook users, prevent or resolve errors by upgrading your Apex classes to the latest available version of the Salesforce API. In particular, the Apex class that inserts the post must be version 32 or higher.

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

If you alter the default Apex class, be sure to select your new Apex class on the setup page, where you can also see Apex processing errors. From Setup, enter *Social Media* in the Quick Find box, then select **Settings**. An email is sent to the administrator when there are errors and, in most circumstances, the data is saved and can be reprocessed. If too many errors are waiting for reprocessing, the Salesforce Social Hub rules are automatically paused to ensure social content is not missed.

We have provided tests for the default Apex class. If you alter your Apex class, you must alter the tests accordingly.

Note: Social personas created after the Summer '15 release have a field indicating which social network created the persona: Source App. This field is set on creation and is not updateable. If your organization uses custom Apex, update it to use this field. Keep in mind that personas created before the Summer 15 release do not have the field. Also, every time new fields are added to the social action you must update your Apex version or the new fields aren't saved.

To create an Apex class, in Setup, enter *Apex Classes* in the Quick Find box, then select **Apex Classes**. You can use the following code to:

- Support person accounts
- Designate a default account ID
- Change the number of days before closed cases are reopened

```
global class MyInboundSocialPostHandlerImpl extends
Social.InboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler {
    global override SObject createPersonaParent(SocialPersona persona) {
        String name = persona.Name;
        if (persona.RealName != null && String.isNotBlank(persona.RealName))
        name = persona.RealName;

        String firstName = '';
        String lastName = 'unknown';
        if (name != null && String.isNotBlank(name)) {
        firstName = name.substringBeforeLast(' ');
        lastName = name.substringAfterLast(' ');
        if (lastName == null || String.isBlank(lastName))
        lastName = firstName;
        }

        //You must have a default Person Account record type
        Account acct = new Account (LastName = lastName, FirstName = firstName);
```

```
        insert acct;
        return acct;
        }

    global override String getDefaultAccountId() {
        return '<account ID>';
    }

    global override Integer getMaxNumberOfDaysClosedToReopenCase() {
        return 5;
    }
}
```

You can use the following code to implement your own social customer service process.

```
global class MyInboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler {
    global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
    SocialPersona persona, Map<String,Object> data) {
        Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();

        // Custom process

        return result;
    }
}
```

The default Apex class sets the contact as the persona parent. To set the persona parent as an account, person account, or lead, create a method to override the persona parent.

If you want a post to go to the error queue, so errors are not lost, your custom apex must do one of two things.

1. Bubble up an exception (recommended).

   ```
   global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
   SocialPersona persona, Map<String, Object> rawData) {
   Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
   result.setSuccess(true);

   try{
   //handle the post here
   } catch(Exception e){
   //log exception, etc
   throw e;
   }
   return result;
   }
   ```

   OR

2. Set the success flag on the response object to false.

   ```
   global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
   SocialPersona persona, Map<String, Object> rawData) {
   Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
   result.setSuccess(true);

   try{
   ```

```
//handle the post here
} catch(Exception e){
//log exception, etc
result.setSuccess(false);
}
return result;
}
```

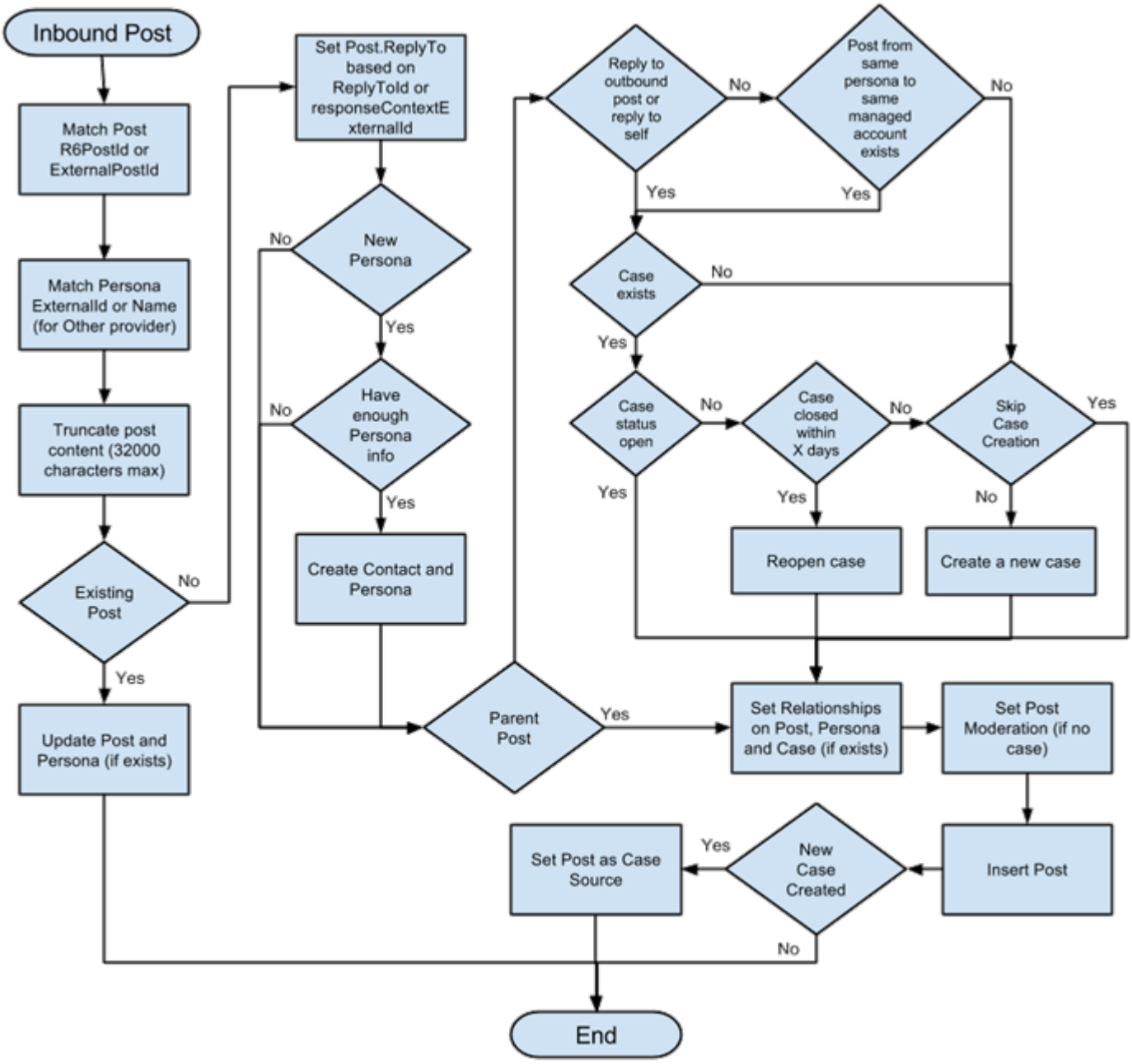
Default Apex Class Process

A visual diagram of an inbound post's path through the default apex class.

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.


Inbound Post
Match Post R6PostId or ExternalPostId
Match Persona ExternalId or Name (for Other provider)
Truncate post content (32000 characters max)
Existing Post
No
Yes
Update Post and Persona (if exists)
Set Post.ReplyTo based on ReplyToId or responseContextExternalId
New Persona
No
Yes
Have enough Persona info
No
Yes
Create Contact and Persona
Parent Post
Yes
Reply to outbound post or reply to self
No
Yes
Post from same persona to same managed account exists
No
Yes
Case exists
No
Yes
Case status open
No
Yes
Case closed within X days
No
Yes
Skip Case Creation
Yes
No
Reopen case
Create a new case
Set Relationships on Post, Persona and Case (if exists)
Set Post Moderation (if no case)
Insert Post
New Case Created
Yes
No
Set Post as Case Source
End

Default Apex Class Reference

Social Customer Service's full default Apex class code. The following Apex class is current as of the Summer '16 release.

For previous versions, see Default Apex Class History on page 30

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

```
global virtual class InboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler
 {

    final static Integer CONTENT_MAX_LENGTH = 32000;
    Boolean isNewCaseCreated = false;

    // Reopen case if it has not been closed for more than this number
    global virtual Integer getMaxNumberOfDaysClosedToReopenCase() {
        return 5;
    }

    // Create a case if one of these post tags are on the SocialPost, regardless of the
skipCreateCase indicator.
    global virtual Set<String> getPostTagsThatCreateCase(){
        return new Set<String>();
    }

    global virtual String getDefaultAccountId() {
        return null;
    }

    global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
SocialPersona persona, Map<String, Object> rawData) {
        Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
        result.setSuccess(true);
        matchPost(post);
        matchPersona(persona);

        if ((post.Content != null) && (post.Content.length() > CONTENT_MAX_LENGTH)) {
            post.Content = post.Content.abbreviate(CONTENT_MAX_LENGTH);
        }

        if (post.Id != null) {
            handleExistingPost(post, persona);
            return result;
        }

        setReplyTo(post, persona);
        buildPersona(persona);
```

```
        Case parentCase = buildParentCase(post, persona, rawData);
        setRelationshipsOnPost(post, persona, parentCase);
        setModeration(post, rawData);

        upsert post;

        if(isNewCaseCreated){
            updateCaseSource(post, parentCase);
        }

        return result;
    }

    private void setModeration(SocialPost post, Map<String, Object> rawData){
     //if we don't automatically create a case, we should flag the post as requiring
moderator review.
     if(post.parentId == null && !isUnsentParent(rawData))
   post.reviewedStatus = 'Needed';
 }

    private void updateCaseSource(SocialPost post, Case parentCase){
        if(parentCase != null) {
            parentCase.SourceId = post.Id;
           //update as a new sobject to prevent undoing any changes done by insert triggers

            update new Case(Id = parentCase.Id, SourceId = parentCase.SourceId);
        }

    }

    private void handleExistingPost(SocialPost post, SocialPersona persona) {
        update post;
        if (persona.id != null)
            updatePersona(persona);
    }

    private void setReplyTo(SocialPost post, SocialPersona persona) {
        SocialPost replyTo = findReplyTo(post, persona);
        if(replyTo.id != null) {
            post.replyToId = replyTo.id;
            post.replyTo = replyTo;
        }
    }

    private SocialPersona buildPersona(SocialPersona persona) {
        if (persona.Id == null)
            createPersona(persona);
        else
            updatePersona(persona);

        return persona;
    }

    private void updatePersona(SocialPersona persona) {
```

```
        try{
            update persona;
        }catch(Exception e) {
            System.debug('Error updating social persona: ' + e.getMessage());
        }
    }

   private Case buildParentCase(SocialPost post, SocialPersona persona, Map<String, Object>
 rawData){
        if(!isUnsentParent(rawData)) {
            Case parentCase = findParentCase(post, persona);
            if (parentCase != null) {
                if (!parentCase.IsClosed) {
                    return parentCase;
                }
                else if (caseShouldBeReopened(parentCase)) {
                    reopenCase(parentCase);
                    return parentCase;
                }
            }
            if(shouldCreateCase(post, rawData)){
                isNewCaseCreated = true;
                return createCase(post, persona);
            }
        }

        return null;
    }

    private boolean caseShouldBeReopened(Case c){
        return c.id != null && c.isClosed && System.now() <
c.closedDate.addDays(getMaxNumberOfDaysClosedToReopenCase());
    }

    private void setRelationshipsOnPost(SocialPost postToUpdate, SocialPersona persona,
Case parentCase) {
        if (persona.Id != null) {
            postToUpdate.PersonaId = persona.Id;

            if(persona.ParentId.getSObjectType() != SocialPost.sObjectType) {
                postToUpdate.WhoId = persona.ParentId;
            }
        }
        if(parentCase != null) {
            postToUpdate.ParentId = parentCase.Id;
        }
    }

    private Case createCase(SocialPost post, SocialPersona persona) {
        Case newCase = new Case(subject = post.Name);
        if (persona != null && persona.ParentId != null) {
            if (persona.ParentId.getSObjectType() == Contact.sObjectType) {
                newCase.ContactId = persona.ParentId;
            } else if (persona.ParentId.getSObjectType() == Account.sObjectType) {
```

```
                newCase.AccountId = persona.ParentId;
            }
        }
        if (post != null && post.Provider != null) {
            newCase.Origin = post.Provider;
        }
        insert newCase;
        return newCase;
    }

    private Case findParentCase(SocialPost post, SocialPersona persona) {
        Case parentCase = null;
        if (post.ReplyTo != null && !isReplyingToAnotherCustomer(post, persona) &&
!isChat(post)) {
            parentCase = findParentCaseFromPostReply(post);
        }
        if (parentCase == null) {
            parentCase = findParentCaseFromPersona(post, persona);
        }
        return parentCase;
    }

    private boolean isReplyingToAnotherCustomer(SocialPost post, SocialPersona persona){
     return !post.ReplyTo.IsOutbound && post.ReplyTo.PersonaId != persona.Id;
    }

    private boolean isChat(SocialPost post){
     return post.messageType == 'Private' || post.messageType == 'Direct';
    }

    private Case findParentCaseFromPostReply(SocialPost post) {
        if (post.ReplyTo != null && String.isNotBlank(post.ReplyTo.ParentId)) {
            List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE
Id = :post.ReplyTo.ParentId LIMIT 1];
            if(!cases.isEmpty()) {
                return cases[0];
            }
        }
        return null;
    }

    private Case findParentCaseFromPersona(SocialPost post, SocialPersona persona) {
        SocialPost lastestInboundPostWithSamePersonaAndRecipient =
findLatestInboundPostBasedOnPersonaAndRecipient(post, persona);
        if (lastestInboundPostWithSamePersonaAndRecipient != null) {
            List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE
id = :lastestInboundPostWithSamePersonaAndRecipient.parentId LIMIT 1];
            if(!cases.isEmpty()) {
                return cases[0];
            }
        }
        return null;
    }
```

```
    private void reopenCase(Case parentCase) {
        SObject[] status = [SELECT MasterLabel FROM CaseStatus WHERE IsClosed = false AND
 IsDefault = true];
        parentCase.Status = ((CaseStatus)status[0]).MasterLabel;
        update parentCase;
    }

    private void matchPost(SocialPost post) {
            if (post.Id != null) return;

        performR6PostIdCheck(post);

        if (post.Id == null){
            performExternalPostIdCheck(post);
        }
    }


    private void performR6PostIdCheck(SocialPost post){
        if(post.R6PostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE R6PostId =
:post.R6PostId LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
    }


    private void performExternalPostIdCheck(SocialPost post) {
        if (post.provider == 'Facebook' && post.messageType == 'Private') return;
        if (post.provider == null || post.externalPostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE ExternalPostId =
:post.ExternalPostId AND Provider = :post.provider LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
    }


    private SocialPost findReplyTo(SocialPost post, SocialPersona persona) {
        if(post.replyToId != null && post.replyTo == null)
            return findReplyToBasedOnReplyToId(post);
        if(post.responseContextExternalId != null){
            if((post.provider == 'Facebook' && post.messageType == 'Private') ||
(post.provider == 'Twitter' && post.messageType == 'Direct')){
                SocialPost replyTo =
findReplyToBasedOnResponseContextExternalPostIdAndProvider(post);
                if(replyTo.id != null)
                 return replyTo;
            }
            return findReplyToBasedOnExternalPostIdAndProvider(post);
        }
        return new SocialPost();
    }
```

```
    private SocialPost findReplyToBasedOnReplyToId(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE id = :post.replyToId LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findReplyToBasedOnExternalPostIdAndProvider(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND ExternalPostId = :post.responseContextExternalId LIMIT
 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

   private SocialPost findReplyToBasedOnResponseContextExternalPostIdAndProvider(SocialPost
 post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND responseContextExternalId =
:post.responseContextExternalId ORDER BY posted DESC NULLS LAST LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findLatestInboundPostBasedOnPersonaAndRecipient(SocialPost post,
SocialPersona persona) {
        if (persona != null && String.isNotBlank(persona.Id) && post != null &&
String.isNotBlank(post.Recipient)) {
            List<SocialPost> posts = [SELECT Id, ParentId FROM SocialPost WHERE Provider
= :post.provider AND Recipient = :post.Recipient AND PersonaId = :persona.id AND IsOutbound
 = false ORDER BY CreatedDate DESC LIMIT 1];
            if (!posts.isEmpty()) {
                return posts[0];
            }
        }
        return null;
    }

    private void matchPersona(SocialPersona persona) {
        if (persona != null) {
            List<SocialPersona> personaList = new List<SocialPersona>();
            if(persona.Provider != 'Other' && String.isNotBlank(persona.ExternalId)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    Provider = :persona.Provider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
           } else if(persona.Provider == 'Other' && String.isNotBlank(persona.ExternalId)
 && String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
```

```
            } else if(persona.Provider == 'Other' && String.isNotBlank(persona.Name) &&
String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    Name = :persona.Name LIMIT 1];
            }

            if (!personaList.isEmpty()) {
                persona.Id = personaList[0].Id;
                persona.ParentId = personaList[0].ParentId;
            }
        }
    }

    private void createPersona(SocialPersona persona) {
        if (persona == null || String.isNotBlank(persona.Id) ||
!isThereEnoughInformationToCreatePersona(persona))
            return;

        SObject parent = createPersonaParent(persona);
        persona.ParentId = parent.Id;
        insert persona;
    }

    private boolean isThereEnoughInformationToCreatePersona(SocialPersona persona) {
        return String.isNotBlank(persona.Name) &&
               String.isNotBlank(persona.Provider) &&
               String.isNotBlank(persona.MediaProvider);
    }

    private boolean shouldCreateCase(SocialPost post, Map<String, Object> rawData) {
        return !isUnsentParent(rawData) && (!hasSkipCreateCaseIndicator(rawData) ||
hasPostTagsThatCreateCase(post));
    }

    private boolean isUnsentParent(Map<String, Object> rawData) {
        Object unsentParent = rawData.get('unsentParent');
       return unsentParent != null && 'true'.equalsIgnoreCase(String.valueOf(unsentParent));

    }

    private boolean hasSkipCreateCaseIndicator(Map<String, Object> rawData) {
        Object skipCreateCase = rawData.get('skipCreateCase');
        return skipCreateCase != null &&
'true'.equalsIgnoreCase(String.valueOf(skipCreateCase));
    }

    private boolean hasPostTagsThatCreateCase(SocialPost post){
        Set<String> postTags = getPostTags(post);
        postTags.retainAll(getPostTagsThatCreateCase());
        return !postTags.isEmpty();
    }

    private Set<String> getPostTags(SocialPost post){
```

```
        Set<String> postTags = new Set<String>();
        if(post.postTags != null)
            postTags.addAll(post.postTags.split(',', 0));
        return postTags;
    }

    global String getPersonaFirstName(SocialPersona persona) {
        String name = getPersonaName(persona);
        String firstName = '';
        if (name.contains(' ')) {
            firstName = name.substringBeforeLast(' ');
        }
        firstName = firstName.abbreviate(40);
        return firstName;
    }

    global String getPersonaLastName(SocialPersona persona) {
        String name = getPersonaName(persona);
        String lastName = name;
        if (name.contains(' ')) {
            lastName = name.substringAfterLast(' ');
        }
        lastName = lastName.abbreviate(80);
        return lastName;
    }

    private String getPersonaName(SocialPersona persona) {
        String name = persona.Name.trim();
        if (String.isNotBlank(persona.RealName)) {
            name = persona.RealName.trim();
        }
        return name;
    }

    global virtual SObject createPersonaParent(SocialPersona persona) {

        String firstName = getPersonaFirstName(persona);
        String lastName = getPersonaLastName(persona);

        Contact contact = new Contact(LastName = lastName, FirstName = firstName);
        String defaultAccountId = getDefaultAccountId();
        if (defaultAccountId != null)
            contact.AccountId = defaultAccountId;
        insert contact;
        return contact;
    }


}
```

Apex Tests for the Default Apex Class

Social Customer Service's tests for the default Apex class code.

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

```
@isTest
public class InboundSocialPostHandlerImplTest {

    static Map<String, Object> sampleSocialData;
    static Social.InboundSocialPostHandlerImpl handler;

    static {
     handler = new Social.InboundSocialPostHandlerImpl();
        sampleSocialData = getSampleSocialData('1');
    }

    static testMethod void verifyNewRecordCreation() {
        SocialPost post = getSocialPost(sampleSocialData);
        SocialPersona persona = getSocialPersona(sampleSocialData);

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost];

        SocialPersona createdPersona = [SELECT Id, ParentId FROM SocialPersona];
        Contact createdContact = [SELECT Id FROM Contact];
        Case createdCase = [SELECT Id, ContactId FROM Case];

        System.assertEquals(createdPost.PersonaId, createdPersona.Id, 'Post is not linked
 to the Persona.');
      System.assertEquals(createdPost.WhoId, createdPersona.ParentId, 'Post is not linked
 to the Contact');
        System.assertEquals(createdPost.ParentId, createdCase.Id, 'Post is not linked to
the Case.');
        System.assertEquals(createdCase.ContactId, createdContact.Id, 'Contact is not
linked to the Case.');
    }

    static testMethod void matchSocialPostRecord() {
        SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
        insert existingPost;
```

```
        SocialPost post = getSocialPost(sampleSocialData);
        post.R6PostId = existingPost.R6PostId;
        SocialPersona persona = getSocialPersona(sampleSocialData);

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        System.assertEquals(1, [SELECT Id FROM SocialPost].size(), 'There should be only
1 post');
    }

    static testMethod void matchSocialPersonaRecord() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;

     SocialPost post = getSocialPost(sampleSocialData);
        SocialPersona persona = getSocialPersona(sampleSocialData);
        persona.ExternalId = existingPersona.ExternalId;

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost];

        SocialPersona createdPersona = [SELECT Id, ParentId FROM SocialPersona];
        Contact createdContact = [SELECT Id FROM Contact];
        Case createdCase = [SELECT Id, ContactId FROM Case];

        System.assertEquals(createdPost.PersonaId, createdPersona.Id, 'Post is not linked
 to the Persona.');
       System.assertEquals(createdPost.WhoId, createdPersona.ParentId, 'Post is not linked
 to the Contact');
        System.assertEquals(createdPost.ParentId, createdCase.Id, 'Post is not linked to
the Case.');
        System.assertEquals(createdCase.ContactId, createdContact.Id, 'Contact is not
linked to the Case.');
    }

    static testMethod void matchCaseRecord() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;
     Case existingCase = new Case(ContactId = existingContact.Id, Subject = 'Test Case');

     insert existingCase;
     SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
     existingPost.ParentId = existingCase.Id;
```

```
     existingPost.WhoId = existingContact.Id;
     existingPost.PersonaId = existingPersona.Id;
     insert existingPost;

     SocialPost post = getSocialPost(sampleSocialData);
     post.responseContextExternalId = existingPost.ExternalPostId;

        test.startTest();
        handler.handleInboundSocialPost(post, existingPersona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost
WHERE R6PostId = :post.R6PostId];
       System.assertEquals(existingPersona.Id, createdPost.PersonaId, 'Post is not linked
 to the Persona.');
        System.assertEquals(existingContact.Id, createdPost.WhoId, 'Post is not linked to
 the Contact');
        System.assertEquals(existingCase.Id, createdPost.ParentId, 'Post is not linked to
 the Case.');
        System.assertEquals(1, [SELECT Id FROM Case].size(), 'There should only be 1
Case.');
    }

    static testMethod void reopenClosedCase() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;
     Case existingCase = new Case(ContactId = existingContact.Id, Subject = 'Test Case',
Status = 'Closed');
     insert existingCase;
     SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
     existingPost.ParentId = existingCase.Id;
     existingPost.WhoId = existingContact.Id;
     existingPost.PersonaId = existingPersona.Id;
     insert existingPost;

     SocialPost post = getSocialPost(sampleSocialData);
     post.responseContextExternalId = existingPost.ExternalPostId;

        test.startTest();
        handler.handleInboundSocialPost(post, existingPersona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost
WHERE R6PostId = :post.R6PostId];
       System.assertEquals(existingPersona.Id, createdPost.PersonaId, 'Post is not linked
 to the Persona.');
        System.assertEquals(existingContact.Id, createdPost.WhoId, 'Post is not linked to
 the Contact');
        System.assertEquals(existingCase.Id, createdPost.ParentId, 'Post is not linked to
 the Case.');
        System.assertEquals(1, [SELECT Id FROM Case].size(), 'There should only be 1
```

```
Case.');
        System.assertEquals(false, [SELECT Id, IsClosed FROM Case WHERE Id =
:existingCase.Id].IsClosed, 'Case should be open.');
    }

    static SocialPost getSocialPost(Map<String, Object> socialData) {
        SocialPost post = new SocialPost();
        post.Name = String.valueOf(socialData.get('source'));
        post.Content = String.valueOf(socialData.get('content'));
        post.Posted = Date.valueOf(String.valueOf(socialData.get('postDate')));
        post.PostUrl = String.valueOf(socialData.get('postUrl'));
        post.Provider = String.valueOf(socialData.get('mediaProvider'));
        post.MessageType = String.valueOf(socialData.get('messageType'));
        post.ExternalPostId = String.valueOf(socialData.get('externalPostId'));
        post.R6PostId = String.valueOf(socialData.get('r6PostId'));
        return post;
    }

    static SocialPersona getSocialPersona(Map<String, Object> socialData) {
        SocialPersona persona = new SocialPersona();
        persona.Name = String.valueOf(socialData.get('author'));
        persona.RealName = String.valueOf(socialData.get('realName'));
        persona.Provider = String.valueOf(socialData.get('mediaProvider'));
        persona.MediaProvider = String.valueOf(socialData.get('mediaProvider'));
        persona.ExternalId = String.valueOf(socialData.get('externalUserId'));
        return persona;
    }

    static Map<String, Object> getSampleSocialData(String suffix) {
        Map<String, Object> socialData = new Map<String, Object>();
        socialData.put('r6PostId', 'R6PostId' + suffix);
        socialData.put('r6SourceId', 'R6SourceId' + suffix);
        socialData.put('postTags', null);
        socialData.put('externalPostId', 'ExternalPostId' + suffix);
        socialData.put('content', 'Content' + suffix);
        socialData.put('postDate', '2015-01-12T12:12:12Z');
        socialData.put('mediaType', 'Twitter');
        socialData.put('author', 'Author');
        socialData.put('skipCreateCase', false);
        socialData.put('mediaProvider', 'TWITTER');
        socialData.put('externalUserId', 'ExternalUserId');
        socialData.put('postUrl', 'PostUrl' + suffix);
        socialData.put('messageType', 'Tweet');
        socialData.put('source', 'Source' + suffix);
        socialData.put('replyToExternalPostId', null);
        socialData.put('realName', 'Real Name');
        return socialData;
    }
}
```

Data Populated into Social Objects

Details on which fields exist in the standard objects, Social Post and Social Persona, and which fields are currently populated by data from Social Studio.

When Social Studio is configured to work with Social Customer Service (SCS), Social Studio sends data to Salesforce in raw format, which is then decoded by the SCS data intake system and appended to two standard Salesforce objects: Social Post and Social Persona. Social Post contains information that is post specific (posts in this context encompass tweets, Twitter direct messages, Facebook posts, comments, comment replies, etc.). Social Persona stores social identity information gleaned from the author information on posts received by SCS.

Note: If you've modified the default Apex class, you may experience alternate mappings.

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

Social Post

The following fields exist on the Social Post object.

Table 1: Social Post Fields

Salesforce Field	Data Value from Social Studio	Sample Data	Notes
AssignedTo	assignedTo	"Joe Smith" (user in Social Studio, not Salesforce)	Not updated
Analyzer Score	analyzerScore	5	Score set on a post in the R6 platform
Attachment Type	mediaUrls array	Image, Video	Populated by SCS when new data arrives in Salesforce - only the first attachment is mapped
Attachment URL	mediaUrls array	http://some.domain/image.jpg	Populated by SCS when new data arrives in Salesforce - only the first attachment is mapped.
Classification	classification	[Custom value]	Populated as admin defines
CommentCount	commentCount	N/A	Not updated
Content	content	Apple teases the new Mac Pro, what do you think	The actual content of the Social post
ExternalPostId	externalPostId	1111222233334444	Native Social Network Id
Handle	author	thehotclothes	N/A
HarvestDate	harvestDate	2013-06-11T13:07:00Z	Date post collected by Social Studio
Headline	source	N/A	Not updated
Id	salesforcePostId	12345678912345	Populated within Salesforce

Salesforce Field	Data Value from Social Studio	Sample Data	Notes
InboundLinkCount	inboundLinkCount	N/A	Not updated
IsOutbound	N/A	Yes/No	Populated within Salesforce
KeywordGroupName	keywordGroupName	N/A	Not updated
Language	language	English	Populated in SCS
LikesAndVotes	likesAndVotes	N/A	Not updated
MediaProvider	mediaProvider	TWITTER	Social network
MediaType	mediaType	Twitter	Social network
MessageType	messageType	Tweet	Possible values: • Twitter: Tweet, Reply, Direct • Facebook: Post, Comment, Reply, Private
Name	source	TWEET FROM: mysamplehandle	System generated by Social Studio.
OutboundSocialAccount	N/A	Northern Trails Outfitters	Populated with Social Account used to publish - only for outbound posts
Parent	N/A	00001728 (linked)	Populated with parent case number if Post associated with case
Persona	N/A	Sample Persona	Populated with author Social Persona if one exists
Posted	postDate	2013-06-11T13:07:00Z	Date-time published on social network.
PostPriority	postPriority	High	Priority set within Social Studio.
PostTags	postTags	post tag 1, post tag 2	Tags are comma-separated.
PostUrl	postUrl	http://twitter.com/mysamplehandle/status/11112222333444	Link to source post
Provider	mediaProvider	Twitter	Set to social network.
R6PostId	r6PostId	12345678	Native Social Studio post ID.
R6SourceId	r6SourceId	1234	Native Social Studio ID for author.
R6TopicId	r6TopicId	1234567890	Native Social Studio ID for either topic profile or managed account

Salesforce Field	Data Value from Social Studio	Sample Data	Notes
Recipient	recipientId	12345678912345	Native ID of recipient on social network
RecipientType	recipientType	Person	N/A
ReplyTo	N/A	Another Social Post (linked)	Dynamically filled by Salesforce logic based on replyToExternalPostId from Social Studio
Sentiment	sentiment	Neutral	N/A
Shares	shares	N/A	Not updated
SourceTags	sourceTags	source tag 1, source tag 2	Source tags used to track types of authors
SpamRating	spamRating	NotSpam	N/A
Status	status	N/A	Not updated
StatusMessage	statusMessage	N/A	Not updated
ThreadSize	threadSize	N/A	Not updated
TopicProfileName	topicProfileName	@my_handle	Name of TP in Social Studio.sd
TopicType	topicType	Keyword | Managed	Whether a topic profile or managed account.
UniqueCommentors	uniqueCommentors	N/A	Not updated
ViewCount	viewCount	N/A	Not updated
Who	N/A	Polymorphic relationship	Can be several other types of records, including Lead. Linked.

Social Persona

The following fields exist on the Social Persona object.

Note: The Social Persona object is only updated when you get a post from someone with an existing persona record. Social Persona is not updated via a parallel process.

Table 2: Social Persona Fields

Salesforce Field	Data Value from Social Studio	Sample Data	Notes
AreWeFollowing	areWeFollowing	N/A	Not updated
Bio	bio	Sample Twitter biography	N/A
ExternalId	externalUserId	1234567890	N/A

Salesforce Field	Data Value from Social Studio	Sample Data	Notes
ExternalPictureURL	profileIconUrl	http://some.domain/image.jpg	N/A
Followers	followers	290	N/A
Following	following	116	N/A
IsBlacklisted	isBlacklisted	N/A	Not updated
IsDefault	N/A	true/false	This value specifies if this record is used to get the avatar image that is displayed on the contact/account. Its used by Social Contacts.N/A
IsFollowingUs	isFollowingUs	N/A	Not updated
Klout	kloutScore	N/A	Not updated
ListedCount	listed	4	N/A
MediaProvider	mediaProvider	Twitter, Facebook etc.	Social network of profile
MediaType	mediaType	Twitter	N/A
Name	author	Joe Smith	N/A
NumberOfFriends	friends	N/A	Not updated
NumberOfTweets	tweets	59546	N/A
Parent	N/A	Contact Name (linked)	Social Persona by default parents to a contact.
ProfileType	authorType	Person	N/A
ProfileUrl	profileUrl	http://twitter.com/mysamplehandle	N/A
Provider	mediaProvider	other	Similar to mediaType but allows fewer values.
R6SourceId	r6SourceId	123456789	Native ID for author
RealName	realName	Joe Smith	N/A
TopicType	topicType	Keyword or Managed	N/A

Additional Data From Social Studio

In addition to the data noted above, certain fields come in the raw data from Social Studio but are not automatically mapped to fields within the Social Post and Social Persona objects. You can access these fields through Visualforce or Apex.

Table 3: Additional Data Fields from Social Studio

Raw Data Field	Notes
authorTags	String
classifiers	Classifier[]
createLead	Boolean
firstName	String
jobId	String
lastName	String
mediaUrls	Raw data comes through as an array of all attachments. SCS matches the first attachment with a known type (image|video) to SocialPost.AttachmentType and SocialPost.AttachmentURL
originalAvatar	String
originalFullName	String
originalScreenName	String
origins	String
privacy	String
r6ParentPostId	Long
recipientId	String
replyToExternalPostId	Raw data used to look up 'In Reply To' Social Post but field not directly written into Social Post
skipCreateCase	Used for the moderation feature introduced in the Summer '14 release; if Yes, SCS skips case creation in the default logic. This field can also be used in customer-specific logic

Default Apex Class History

Social Customer Service's full default Apex class for prior releases.

For the current release, see Default Apex Class Reference on page 14

EDITIONS

Available in: Salesforce Classic

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

Default Apex Class and Test for Spring '16

```
global virtual class InboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler
 {

    final static Integer CONTENT_MAX_LENGTH = 32000;
    Boolean isNewCaseCreated = false;

    // Reopen case if it has not been closed for more than this number
    global virtual Integer getMaxNumberOfDaysClosedToReopenCase() {
        return 5;
    }

    // Create a case if one of these post tags are on the SocialPost, regardless of the
skipCreateCase indicator.
    global virtual Set<String> getPostTagsThatCreateCase(){
        return new Set<String>();
    }

    global virtual String getDefaultAccountId() {
        return null;
    }

    global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
SocialPersona persona, Map<String, Object> rawData) {
        Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
        result.setSuccess(true);
        matchPost(post);
        matchPersona(persona);

        if ((post.Content != null) && (post.Content.length() > CONTENT_MAX_LENGTH)) {
            post.Content = post.Content.abbreviate(CONTENT_MAX_LENGTH);
        }

        if (post.Id != null) {
            handleExistingPost(post, persona);
            return result;
        }

        setReplyTo(post, persona);
        buildPersona(persona);
        Case parentCase = buildParentCase(post, persona, rawData);
        setRelationshipsOnPost(post, persona, parentCase);
        setModeration(post);

        upsert post;

        if(isNewCaseCreated){
            updateCaseSource(post, parentCase);
        }

        return result;
    }
```

```
    private void setModeration(SocialPost post){
     //if we don't automatically create a case, we should flag the post as requiring
moderator review.
     if(post.parentId == null)
   post.reviewedStatus = 'Needed';
 }

    private void updateCaseSource(SocialPost post, Case parentCase){
        if(parentCase != null) {
            parentCase.SourceId = post.Id;
            update parentCase;
        }

    }

    private void handleExistingPost(SocialPost post, SocialPersona persona) {
        update post;
        if (persona.id != null)
            updatePersona(persona);
    }

    private void setReplyTo(SocialPost post, SocialPersona persona) {
        SocialPost replyTo = findReplyTo(post, persona);
        if(replyTo.id != null) {
            post.replyToId = replyTo.id;
            post.replyTo = replyTo;
        }
    }

    private SocialPersona buildPersona(SocialPersona persona) {
        if (persona.Id == null)
            createPersona(persona);
        else
            updatePersona(persona);

        return persona;
    }

    private void updatePersona(SocialPersona persona) {
        try{
            update persona;
        }catch(Exception e) {
            System.debug('Error updating social persona: ' + e.getMessage());
        }
    }

   private Case buildParentCase(SocialPost post, SocialPersona persona, Map<String, Object>
 rawData){
        Case parentCase = findParentCase(post, persona);
        if (parentCase != null) {
            if (!parentCase.IsClosed) {
                return parentCase;
            }
            else if (caseShouldBeReopened(parentCase)) {
```

```
                reopenCase(parentCase);
                return parentCase;
            }
        }
        if(shouldCreateCase(post, rawData)){
            isNewCaseCreated = true;
            return createCase(post, persona);
        }

        return null;
    }

    private boolean caseShouldBeReopened(Case c){
        return c.id != null && c.isClosed && System.now() <
c.closedDate.addDays(getMaxNumberOfDaysClosedToReopenCase());
    }

    private void setRelationshipsOnPost(SocialPost postToUpdate, SocialPersona persona,
Case parentCase) {
        if (persona.Id != null) {
            postToUpdate.PersonaId = persona.Id;

            if(persona.ParentId.getSObjectType() != SocialPost.sObjectType) {
                postToUpdate.WhoId = persona.ParentId;
            }
        }
        if(parentCase != null) {
            postToUpdate.ParentId = parentCase.Id;
        }
    }

    private Case createCase(SocialPost post, SocialPersona persona) {
        Case newCase = new Case(subject = post.Name);
        if (persona != null && persona.ParentId != null) {
            if (persona.ParentId.getSObjectType() == Contact.sObjectType) {
                newCase.ContactId = persona.ParentId;
            } else if (persona.ParentId.getSObjectType() == Account.sObjectType) {
                newCase.AccountId = persona.ParentId;
            }
        }
        if (post != null && post.Provider != null) {
            newCase.Origin = post.Provider;
        }
        insert newCase;
        return newCase;
    }

    private Case findParentCase(SocialPost post, SocialPersona persona) {
        Case parentCase = null;
        if (post.ReplyTo != null && !isReplyingToAnotherCustomer(post, persona) &&
!isChat(post)) {
            parentCase = findParentCaseFromPostReply(post);
        }
        if (parentCase == null) {
```

```
            parentCase = findParentCaseFromPersona(post, persona);
        }
        return parentCase;
    }

    private boolean isReplyingToAnotherCustomer(SocialPost post, SocialPersona persona){
     return !post.ReplyTo.IsOutbound && post.ReplyTo.PersonaId != persona.Id;
    }

    private boolean isChat(SocialPost post){
     return post.messageType == 'Private' || post.messageType == 'Direct';
    }

    private Case findParentCaseFromPostReply(SocialPost post) {
        if (post.ReplyTo != null && String.isNotBlank(post.ReplyTo.ParentId)) {
            List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE
Id = :post.ReplyTo.ParentId LIMIT 1];
            if(!cases.isEmpty()) {
                return cases[0];
            }
        }
        return null;
    }

    private Case findParentCaseFromPersona(SocialPost post, SocialPersona persona) {
        SocialPost lastestInboundPostWithSamePersonaAndRecipient =
findLatestInboundPostBasedOnPersonaAndRecipient(post, persona);
        if (lastestInboundPostWithSamePersonaAndRecipient != null) {
            List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE
id = :lastestInboundPostWithSamePersonaAndRecipient.parentId LIMIT 1];
            if(!cases.isEmpty()) {
                return cases[0];
            }
        }
        return null;
    }

    private void reopenCase(Case parentCase) {
        SObject[] status = [SELECT MasterLabel FROM CaseStatus WHERE IsClosed = false AND
 IsDefault = true];
        parentCase.Status = ((CaseStatus)status[0]).MasterLabel;
        update parentCase;
    }

    private void matchPost(SocialPost post) {
            if (post.Id != null) return;

        performR6PostIdCheck(post);

        if (post.Id == null){
            performExternalPostIdCheck(post);
        }
    }
```

```
    private void performR6PostIdCheck(SocialPost post){
        if(post.R6PostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE R6PostId =
:post.R6PostId LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
    }


    private void performExternalPostIdCheck(SocialPost post) {
        if (post.provider == 'Facebook' && post.messageType == 'Private') return;
        if (post.provider == null || post.externalPostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE ExternalPostId =
:post.ExternalPostId AND Provider = :post.provider LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
    }


    private SocialPost findReplyTo(SocialPost post, SocialPersona persona) {
        if(post.replyToId != null && post.replyTo == null)
            return findReplyToBasedOnReplyToId(post);
        if(post.responseContextExternalId != null){
            if((post.provider == 'Facebook' && post.messageType == 'Private') ||
(post.provider == 'Twitter' && post.messageType == 'Direct')){
                SocialPost replyTo =
findReplyToBasedOnResponseContextExternalPostIdAndProvider(post);
                if(replyTo.id != null)
                 return replyTo;
            }
            return findReplyToBasedOnExternalPostIdAndProvider(post);
        }
        return new SocialPost();
    }

    private SocialPost findReplyToBasedOnReplyToId(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE id = :post.replyToId LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findReplyToBasedOnExternalPostIdAndProvider(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND ExternalPostId = :post.responseContextExternalId LIMIT
 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }
```

```
   private SocialPost findReplyToBasedOnResponseContextExternalPostIdAndProvider(SocialPost
 post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND responseContextExternalId =
:post.responseContextExternalId ORDER BY posted DESC NULLS LAST LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findLatestInboundPostBasedOnPersonaAndRecipient(SocialPost post,
SocialPersona persona) {
        if (persona != null && String.isNotBlank(persona.Id) && post != null &&
String.isNotBlank(post.Recipient)) {
            List<SocialPost> posts = [SELECT Id, ParentId FROM SocialPost WHERE Provider
= :post.provider AND Recipient = :post.Recipient AND PersonaId = :persona.id AND IsOutbound
 = false ORDER BY CreatedDate DESC LIMIT 1];
            if (!posts.isEmpty()) {
                return posts[0];
            }
        }
        return null;
    }

    private void matchPersona(SocialPersona persona) {
        if (persona != null) {
            List<SocialPersona> personaList = new List<SocialPersona>();
            if(persona.Provider != 'Other' && String.isNotBlank(persona.ExternalId)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    Provider = :persona.Provider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
           } else if(persona.Provider == 'Other' && String.isNotBlank(persona.ExternalId)
 && String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
            } else if(persona.Provider == 'Other' && String.isNotBlank(persona.Name) &&
String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    Name = :persona.Name LIMIT 1];
            }

            if (!personaList.isEmpty()) {
                persona.Id = personaList[0].Id;
                persona.ParentId = personaList[0].ParentId;
            }
        }
    }

    private void createPersona(SocialPersona persona) {
        if (persona == null || String.isNotBlank(persona.Id) ||
!isThereEnoughInformationToCreatePersona(persona))
```

```
            return;

        SObject parent = createPersonaParent(persona);
        persona.ParentId = parent.Id;
        insert persona;
    }

    private boolean isThereEnoughInformationToCreatePersona(SocialPersona persona) {
        return String.isNotBlank(persona.Name) &&
               String.isNotBlank(persona.Provider) &&
               String.isNotBlank(persona.MediaProvider);
    }

    private boolean shouldCreateCase(SocialPost post, Map<String, Object> rawData){
        return !hasSkipCreateCaseIndicator(rawData) || hasPostTagsThatCreateCase(post);
    }

    private boolean hasSkipCreateCaseIndicator(Map<String, Object> rawData) {
        Object skipCreateCase = rawData.get('skipCreateCase');
        return skipCreateCase != null &&
'true'.equalsIgnoreCase(String.valueOf(skipCreateCase));
    }

    private boolean hasPostTagsThatCreateCase(SocialPost post){
        Set<String> postTags = getPostTags(post);
        postTags.retainAll(getPostTagsThatCreateCase());
        return !postTags.isEmpty();
    }

    private Set<String> getPostTags(SocialPost post){
        Set<String> postTags = new Set<String>();
        if(post.postTags != null)
            postTags.addAll(post.postTags.split(',', 0));
        return postTags;
    }

    global String getPersonaFirstName(SocialPersona persona) {
        String name = getPersonaName(persona);
        String firstName = '';
        if (name.contains(' ')) {
            firstName = name.substringBeforeLast(' ');
        }
        firstName = firstName.abbreviate(40);
        return firstName;
    }

    global String getPersonaLastName(SocialPersona persona) {
        String name = getPersonaName(persona);
        String lastName = name;
        if (name.contains(' ')) {
            lastName = name.substringAfterLast(' ');
        }
        lastName = lastName.abbreviate(80);
        return lastName;
```

```
    }

    private String getPersonaName(SocialPersona persona) {
        String name = persona.Name.trim();
        if (String.isNotBlank(persona.RealName)) {
            name = persona.RealName.trim();
        }
        return name;
    }

    global virtual SObject createPersonaParent(SocialPersona persona) {

        String firstName = getPersonaFirstName(persona);
        String lastName = getPersonaLastName(persona);

        Contact contact = new Contact(LastName = lastName, FirstName = firstName);
        String defaultAccountId = getDefaultAccountId();
        if (defaultAccountId != null)
            contact.AccountId = defaultAccountId;
        insert contact;
        return contact;
    }


}
```

Test

```
@isTest
public class InboundSocialPostHandlerImplTest {

    static Map<String, Object> sampleSocialData;
    static Social.InboundSocialPostHandlerImpl handler;

    static {
     handler = new Social.InboundSocialPostHandlerImpl();
        sampleSocialData = getSampleSocialData('1');
    }

    static testMethod void verifyNewRecordCreation() {
        SocialPost post = getSocialPost(sampleSocialData);
        SocialPersona persona = getSocialPersona(sampleSocialData);

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost];

        SocialPersona createdPersona = [SELECT Id, ParentId FROM SocialPersona];
        Contact createdContact = [SELECT Id FROM Contact];
        Case createdCase = [SELECT Id, ContactId FROM Case];

        System.assertEquals(createdPost.PersonaId, createdPersona.Id, 'Post is not linked
 to the Persona.');
```

```
       System.assertEquals(createdPost.WhoId, createdPersona.ParentId, 'Post is not linked
 to the Contact');
        System.assertEquals(createdPost.ParentId, createdCase.Id, 'Post is not linked to
the Case.');
        System.assertEquals(createdCase.ContactId, createdContact.Id, 'Contact is not
linked to the Case.');
    }

    static testMethod void matchSocialPostRecord() {
        SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
        insert existingPost;

        SocialPost post = getSocialPost(sampleSocialData);
        post.R6PostId = existingPost.R6PostId;
        SocialPersona persona = getSocialPersona(sampleSocialData);

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        System.assertEquals(1, [SELECT Id FROM SocialPost].size(), 'There should be only
1 post');
    }

    static testMethod void matchSocialPersonaRecord() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;

     SocialPost post = getSocialPost(sampleSocialData);
        SocialPersona persona = getSocialPersona(sampleSocialData);
        persona.ExternalId = existingPersona.ExternalId;

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost];

        SocialPersona createdPersona = [SELECT Id, ParentId FROM SocialPersona];
        Contact createdContact = [SELECT Id FROM Contact];
        Case createdCase = [SELECT Id, ContactId FROM Case];

        System.assertEquals(createdPost.PersonaId, createdPersona.Id, 'Post is not linked
 to the Persona.');
       System.assertEquals(createdPost.WhoId, createdPersona.ParentId, 'Post is not linked
 to the Contact');
        System.assertEquals(createdPost.ParentId, createdCase.Id, 'Post is not linked to
the Case.');
        System.assertEquals(createdCase.ContactId, createdContact.Id, 'Contact is not
linked to the Case.');
    }
```

```
    static testMethod void matchCaseRecord() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;
     Case existingCase = new Case(ContactId = existingContact.Id, Subject = 'Test Case');

     insert existingCase;
     SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
     existingPost.ParentId = existingCase.Id;
     existingPost.WhoId = existingContact.Id;
     existingPost.PersonaId = existingPersona.Id;
     insert existingPost;

     SocialPost post = getSocialPost(sampleSocialData);
     post.responseContextExternalId = existingPost.ExternalPostId;

        test.startTest();
        handler.handleInboundSocialPost(post, existingPersona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost
WHERE R6PostId = :post.R6PostId];
       System.assertEquals(existingPersona.Id, createdPost.PersonaId, 'Post is not linked
 to the Persona.');
        System.assertEquals(existingContact.Id, createdPost.WhoId, 'Post is not linked to
 the Contact');
        System.assertEquals(existingCase.Id, createdPost.ParentId, 'Post is not linked to
 the Case.');
        System.assertEquals(1, [SELECT Id FROM Case].size(), 'There should only be 1
Case.');
    }

    static testMethod void reopenClosedCase() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;
     Case existingCase = new Case(ContactId = existingContact.Id, Subject = 'Test Case',
Status = 'Closed');
     insert existingCase;
     SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
     existingPost.ParentId = existingCase.Id;
     existingPost.WhoId = existingContact.Id;
     existingPost.PersonaId = existingPersona.Id;
     insert existingPost;

     SocialPost post = getSocialPost(sampleSocialData);
     post.responseContextExternalId = existingPost.ExternalPostId;

        test.startTest();
```

```
        handler.handleInboundSocialPost(post, existingPersona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost
WHERE R6PostId = :post.R6PostId];
       System.assertEquals(existingPersona.Id, createdPost.PersonaId, 'Post is not linked
 to the Persona.');
        System.assertEquals(existingContact.Id, createdPost.WhoId, 'Post is not linked to
 the Contact');
        System.assertEquals(existingCase.Id, createdPost.ParentId, 'Post is not linked to
 the Case.');
        System.assertEquals(1, [SELECT Id FROM Case].size(), 'There should only be 1
Case.');
        System.assertEquals(false, [SELECT Id, IsClosed FROM Case WHERE Id =
:existingCase.Id].IsClosed, 'Case should be open.');
    }

    static SocialPost getSocialPost(Map<String, Object> socialData) {
        SocialPost post = new SocialPost();
        post.Name = String.valueOf(socialData.get('source'));
        post.Content = String.valueOf(socialData.get('content'));
        post.Posted = Date.valueOf(String.valueOf(socialData.get('postDate')));
        post.PostUrl = String.valueOf(socialData.get('postUrl'));
        post.Provider = String.valueOf(socialData.get('mediaProvider'));
        post.MessageType = String.valueOf(socialData.get('messageType'));
        post.ExternalPostId = String.valueOf(socialData.get('externalPostId'));
        post.R6PostId = String.valueOf(socialData.get('r6PostId'));
        return post;
    }

    static SocialPersona getSocialPersona(Map<String, Object> socialData) {
        SocialPersona persona = new SocialPersona();
        persona.Name = String.valueOf(socialData.get('author'));
        persona.RealName = String.valueOf(socialData.get('realName'));
        persona.Provider = String.valueOf(socialData.get('mediaProvider'));
        persona.MediaProvider = String.valueOf(socialData.get('mediaProvider'));
        persona.ExternalId = String.valueOf(socialData.get('externalUserId'));
        return persona;
    }

    static Map<String, Object> getSampleSocialData(String suffix) {
        Map<String, Object> socialData = new Map<String, Object>();
        socialData.put('r6PostId', 'R6PostId' + suffix);
        socialData.put('r6SourceId', 'R6SourceId' + suffix);
        socialData.put('postTags', null);
        socialData.put('externalPostId', 'ExternalPostId' + suffix);
        socialData.put('content', 'Content' + suffix);
        socialData.put('postDate', '2015-01-12T12:12:12Z');
        socialData.put('mediaType', 'Twitter');
        socialData.put('author', 'Author');
        socialData.put('skipCreateCase', false);
        socialData.put('mediaProvider', 'TWITTER');
        socialData.put('externalUserId', 'ExternalUserId');
        socialData.put('postUrl', 'PostUrl' + suffix);
```

```
        socialData.put('messageType', 'Tweet');
        socialData.put('source', 'Source' + suffix);
        socialData.put('replyToExternalPostId', null);
        socialData.put('realName', 'Real Name');
        return socialData;
    }
}
```

Default Apex Class and Test for Winter '15

```
global virtual class InboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler
 {

    final static Integer CONTENT_MAX_LENGTH = 32000;
    Boolean isNewCaseCreated = false;

    // Reopen case if it has not been closed for more than this number
    global virtual Integer getMaxNumberOfDaysClosedToReopenCase() {
        return 5;
    }

    // Create a case if one of these post tags are on the SocialPost, regardless of the
skipCreateCase indicator.
    global virtual Set<String> getPostTagsThatCreateCase(){
        return new Set<String>();
    }

    global virtual String getDefaultAccountId() {
        return null;
    }

    global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
SocialPersona persona, Map<String, Object> rawData) {
        Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
        result.setSuccess(true);
        matchPost(post);
        matchPersona(persona);

        if ((post.Content != null) && (post.Content.length() > CONTENT_MAX_LENGTH)) {
            post.Content = post.Content.abbreviate(CONTENT_MAX_LENGTH);
        }

        if (post.Id != null) {
            handleExistingPost(post, persona);
            return result;
        }

        setReplyTo(post, persona);
        buildPersona(persona);
        Case parentCase = buildParentCase(post, persona, rawData);
        setRelationshipsOnPost(post, persona, parentCase);

        upsert post;
```

```
        if(isNewCaseCreated){
            updateCaseSource(post, parentCase);
        }

        return result;
    }

    private void updateCaseSource(SocialPost post, Case parentCase){
        if(parentCase != null) {
            parentCase.SourceId = post.Id;
            update parentCase;
        }

    }

    private void handleExistingPost(SocialPost post, SocialPersona persona) {
        update post;
        if (persona.id != null)
            updatePersona(persona);
    }

    private void setReplyTo(SocialPost post, SocialPersona persona) {
        SocialPost replyTo = findReplyTo(post, persona);
        if(replyTo.id != null) {
            post.replyToId = replyTo.id;
            post.replyTo = replyTo;
        }
    }

    private SocialPersona buildPersona(SocialPersona persona) {
        if (persona.Id == null)
            createPersona(persona);
        else
            updatePersona(persona);

        return persona;
    }

    private void updatePersona(SocialPersona persona) {
        try{
            update persona;
        }catch(Exception e) {
            System.debug('Error updating social persona: ' + e.getMessage());
        }
    }

   private Case buildParentCase(SocialPost post, SocialPersona persona, Map<String, Object>
rawData){
        Case parentCase = findParentCase(post, persona);
        if (parentCase != null) {
            if (!parentCase.IsClosed) {
                return parentCase;
            }
```

```
            else if (caseShouldBeReopened(parentCase)) {
                reopenCase(parentCase);
                return parentCase;
            }
        }
        if(shouldCreateCase(post, rawData)){
            isNewCaseCreated = true;
            return createCase(post, persona);
        }

        return null;
    }

    private boolean caseShouldBeReopened(Case c){
        return c.id != null && c.isClosed && System.now() <
c.closedDate.addDays(getMaxNumberOfDaysClosedToReopenCase());
    }

    private void setRelationshipsOnPost(SocialPost postToUpdate, SocialPersona persona,
Case parentCase) {
        if (persona.Id != null) {
            postToUpdate.PersonaId = persona.Id;

            if(persona.ParentId.getSObjectType() != SocialPost.sObjectType) {
                postToUpdate.WhoId = persona.ParentId;
            }
        }
        if(parentCase != null) {
            postToUpdate.ParentId = parentCase.Id;
        }
    }

    private Case createCase(SocialPost post, SocialPersona persona) {
        Case newCase = new Case(subject = post.Name);
        if (persona != null && persona.ParentId != null) {
            if (persona.ParentId.getSObjectType() == Contact.sObjectType) {
                newCase.ContactId = persona.ParentId;
            } else if (persona.ParentId.getSObjectType() == Account.sObjectType) {
                newCase.AccountId = persona.ParentId;
            }
        }
        if (post != null && post.Provider != null) {
            newCase.Origin = post.Provider;
        }
        insert newCase;
        return newCase;
    }

    private Case findParentCase(SocialPost post, SocialPersona persona) {
        Case parentCase = null;
        if (post.ReplyTo != null && !isReplyingToAnotherCustomer(post, persona) &&
!isChat(post)) {
            parentCase = findParentCaseFromPostReply(post);
        }
```

```
        if (parentCase == null) {
            parentCase = findParentCaseFromPersona(post, persona);
        }
        return parentCase;
    }

    private boolean isReplyingToAnotherCustomer(SocialPost post, SocialPersona persona){
     return !post.ReplyTo.IsOutbound && post.ReplyTo.PersonaId != persona.Id;
    }

    private boolean isChat(SocialPost post){
     return post.messageType == 'Private' || post.messageType == 'Direct';
    }

    private Case findParentCaseFromPostReply(SocialPost post) {
        if (post.ReplyTo != null && String.isNotBlank(post.ReplyTo.ParentId)) {
            List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE
Id = :post.ReplyTo.ParentId LIMIT 1];
            if(!cases.isEmpty()) {
                return cases[0];
            }
        }
        return null;
    }

    private Case findParentCaseFromPersona(SocialPost post, SocialPersona persona) {
        SocialPost lastestInboundPostWithSamePersonaAndRecipient =
findLatestInboundPostBasedOnPersonaAndRecipient(post, persona);
        if (lastestInboundPostWithSamePersonaAndRecipient != null) {
            List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE
id = :lastestInboundPostWithSamePersonaAndRecipient.parentId LIMIT 1];
            if(!cases.isEmpty()) {
                return cases[0];
            }
        }
        return null;
    }

    private void reopenCase(Case parentCase) {
        SObject[] status = [SELECT MasterLabel FROM CaseStatus WHERE IsClosed = false AND
 IsDefault = true];
        parentCase.Status = ((CaseStatus)status[0]).MasterLabel;
        update parentCase;
    }

    private void matchPost(SocialPost post) {
            if (post.Id != null) return;

        performR6PostIdCheck(post);

        if (post.Id == null){
            performExternalPostIdCheck(post);
        }
    }
```

```
    private void performR6PostIdCheck(SocialPost post){
        if(post.R6PostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE R6PostId =
:post.R6PostId LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
    }


    private void performExternalPostIdCheck(SocialPost post) {
        if (post.provider == 'Facebook' && post.messageType == 'Private') return;
        if (post.provider == null || post.externalPostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE ExternalPostId =
:post.ExternalPostId AND Provider = :post.provider LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
    }


    private SocialPost findReplyTo(SocialPost post, SocialPersona persona) {
        if(post.replyToId != null && post.replyTo == null)
            return findReplyToBasedOnReplyToId(post);
        if(post.responseContextExternalId != null){
            if((post.provider == 'Facebook' && post.messageType == 'Private') ||
(post.provider == 'Twitter' && post.messageType == 'Direct')){
                SocialPost replyTo =
findReplyToBasedOnResponseContextExternalPostIdAndProvider(post);
                if(replyTo.id != null)
                 return replyTo;
            }
            return findReplyToBasedOnExternalPostIdAndProvider(post);
        }
        return new SocialPost();
    }

    private SocialPost findReplyToBasedOnReplyToId(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE id = :post.replyToId LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findReplyToBasedOnExternalPostIdAndProvider(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND ExternalPostId = :post.responseContextExternalId LIMIT
 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
```

```
    }

   private SocialPost findReplyToBasedOnResponseContextExternalPostIdAndProvider(SocialPost
 post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND responseContextExternalId =
:post.responseContextExternalId ORDER BY posted DESC NULLS LAST LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findLatestInboundPostBasedOnPersonaAndRecipient(SocialPost post,
SocialPersona persona) {
        if (persona != null && String.isNotBlank(persona.Id) && post != null &&
String.isNotBlank(post.Recipient)) {
            List<SocialPost> posts = [SELECT Id, ParentId FROM SocialPost WHERE Provider
= :post.provider AND Recipient = :post.Recipient AND PersonaId = :persona.id AND IsOutbound
 = false ORDER BY CreatedDate DESC LIMIT 1];
            if (!posts.isEmpty()) {
                return posts[0];
            }
        }
        return null;
    }

    private void matchPersona(SocialPersona persona) {
        if (persona != null) {
            List<SocialPersona> personaList = new List<SocialPersona>();
            if(persona.Provider != 'Other' && String.isNotBlank(persona.ExternalId)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    Provider = :persona.Provider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
           } else if(persona.Provider == 'Other' && String.isNotBlank(persona.ExternalId)
 && String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
            } else if(persona.Provider == 'Other' && String.isNotBlank(persona.Name) &&
String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    Name = :persona.Name LIMIT 1];
            }

            if (!personaList.isEmpty()) {
                persona.Id = personaList[0].Id;
                persona.ParentId = personaList[0].ParentId;
            }
        }
    }

    private void createPersona(SocialPersona persona) {
        if (persona == null || String.isNotBlank(persona.Id) ||
```

```
!isThereEnoughInformationToCreatePersona(persona))
            return;

        SObject parent = createPersonaParent(persona);
        persona.ParentId = parent.Id;
        insert persona;
    }

    private boolean isThereEnoughInformationToCreatePersona(SocialPersona persona) {
        return String.isNotBlank(persona.Name) &&
               String.isNotBlank(persona.Provider) &&
               String.isNotBlank(persona.MediaProvider);
    }

    private boolean shouldCreateCase(SocialPost post, Map<String, Object> rawData){
        return !hasSkipCreateCaseIndicator(rawData) || hasPostTagsThatCreateCase(post);
    }

    private boolean hasSkipCreateCaseIndicator(Map<String, Object> rawData) {
        Object skipCreateCase = rawData.get('skipCreateCase');
        return skipCreateCase != null &&
'true'.equalsIgnoreCase(String.valueOf(skipCreateCase));
    }

    private boolean hasPostTagsThatCreateCase(SocialPost post){
        Set<String> postTags = getPostTags(post);
        postTags.retainAll(getPostTagsThatCreateCase());
        return !postTags.isEmpty();
    }

    private Set<String> getPostTags(SocialPost post){
        Set<String> postTags = new Set<String>();
        if(post.postTags != null)
            postTags.addAll(post.postTags.split(',', 0));
        return postTags;
    }

    global String getPersonaFirstName(SocialPersona persona) {
        String name = getPersonaName(persona);
        String firstName = '';
        if (name.contains(' ')) {
            firstName = name.substringBeforeLast(' ');
        }
        firstName = firstName.abbreviate(40);
        return firstName;
    }

    global String getPersonaLastName(SocialPersona persona) {
        String name = getPersonaName(persona);
        String lastName = name;
        if (name.contains(' ')) {
            lastName = name.substringAfterLast(' ');
        }
        lastName = lastName.abbreviate(80);
```

```
        return lastName;
    }

    private String getPersonaName(SocialPersona persona) {
        String name = persona.Name.trim();
        if (String.isNotBlank(persona.RealName)) {
            name = persona.RealName.trim();
        }
        return name;
    }

    global virtual SObject createPersonaParent(SocialPersona persona) {

        String firstName = getPersonaFirstName(persona);
        String lastName = getPersonaLastName(persona);

        Contact contact = new Contact(LastName = lastName, FirstName = firstName);
        String defaultAccountId = getDefaultAccountId();
        if (defaultAccountId != null)
            contact.AccountId = defaultAccountId;
        insert contact;
        return contact;
    }


}
```

Test

```
@isTest
public class InboundSocialPostHandlerImplTest {

    static Map<String, Object> sampleSocialData;
    static Social.InboundSocialPostHandlerImpl handler;

    static {
     handler = new Social.InboundSocialPostHandlerImpl();
        sampleSocialData = getSampleSocialData('1');
    }

    static testMethod void verifyNewRecordCreation() {
        SocialPost post = getSocialPost(sampleSocialData);
        SocialPersona persona = getSocialPersona(sampleSocialData);

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost];

        SocialPersona createdPersona = [SELECT Id, ParentId FROM SocialPersona];
        Contact createdContact = [SELECT Id FROM Contact];
        Case createdCase = [SELECT Id, ContactId FROM Case];

        System.assertEquals(createdPost.PersonaId, createdPersona.Id, 'Post is not linked
```

```
 to the Persona.');
       System.assertEquals(createdPost.WhoId, createdPersona.ParentId, 'Post is not linked
 to the Contact');
        System.assertEquals(createdPost.ParentId, createdCase.Id, 'Post is not linked to
the Case.');
        System.assertEquals(createdCase.ContactId, createdContact.Id, 'Contact is not
linked to the Case.');
    }

    static testMethod void matchSocialPostRecord() {
        SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
        insert existingPost;

        SocialPost post = getSocialPost(sampleSocialData);
        post.R6PostId = existingPost.R6PostId;
        SocialPersona persona = getSocialPersona(sampleSocialData);

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        System.assertEquals(1, [SELECT Id FROM SocialPost].size(), 'There should be only
1 post');
    }

    static testMethod void matchSocialPersonaRecord() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;

     SocialPost post = getSocialPost(sampleSocialData);
        SocialPersona persona = getSocialPersona(sampleSocialData);
        persona.ExternalId = existingPersona.ExternalId;

        test.startTest();
        handler.handleInboundSocialPost(post, persona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost];

        SocialPersona createdPersona = [SELECT Id, ParentId FROM SocialPersona];
        Contact createdContact = [SELECT Id FROM Contact];
        Case createdCase = [SELECT Id, ContactId FROM Case];

        System.assertEquals(createdPost.PersonaId, createdPersona.Id, 'Post is not linked
 to the Persona.');
       System.assertEquals(createdPost.WhoId, createdPersona.ParentId, 'Post is not linked
 to the Contact');
        System.assertEquals(createdPost.ParentId, createdCase.Id, 'Post is not linked to
the Case.');
        System.assertEquals(createdCase.ContactId, createdContact.Id, 'Contact is not
linked to the Case.');
```

```
    }

    static testMethod void matchCaseRecord() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;
     Case existingCase = new Case(ContactId = existingContact.Id, Subject = 'Test Case');

     insert existingCase;
     SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
     existingPost.ParentId = existingCase.Id;
     existingPost.WhoId = existingContact.Id;
     existingPost.PersonaId = existingPersona.Id;
     insert existingPost;

     SocialPost post = getSocialPost(sampleSocialData);
     post.responseContextExternalId = existingPost.ExternalPostId;

        test.startTest();
        handler.handleInboundSocialPost(post, existingPersona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost
WHERE R6PostId = :post.R6PostId];
       System.assertEquals(existingPersona.Id, createdPost.PersonaId, 'Post is not linked
 to the Persona.');
        System.assertEquals(existingContact.Id, createdPost.WhoId, 'Post is not linked to
 the Contact');
        System.assertEquals(existingCase.Id, createdPost.ParentId, 'Post is not linked to
 the Case.');
        System.assertEquals(1, [SELECT Id FROM Case].size(), 'There should only be 1
Case.');
    }

    static testMethod void reopenClosedCase() {
     Contact existingContact = new Contact(LastName = 'LastName');
     insert existingContact;
     SocialPersona existingPersona = getSocialPersona(getSampleSocialData('2'));
     existingPersona.ParentId = existingContact.Id;
     insert existingPersona;
     Case existingCase = new Case(ContactId = existingContact.Id, Subject = 'Test Case',
Status = 'Closed');
     insert existingCase;
     SocialPost existingPost = getSocialPost(getSampleSocialData('2'));
     existingPost.ParentId = existingCase.Id;
     existingPost.WhoId = existingContact.Id;
     existingPost.PersonaId = existingPersona.Id;
     insert existingPost;

     SocialPost post = getSocialPost(sampleSocialData);
     post.responseContextExternalId = existingPost.ExternalPostId;
```

```
        test.startTest();
        handler.handleInboundSocialPost(post, existingPersona, sampleSocialData);
        test.stopTest();

        SocialPost createdPost = [SELECT Id, PersonaId, ParentId, WhoId FROM SocialPost
WHERE R6PostId = :post.R6PostId];
       System.assertEquals(existingPersona.Id, createdPost.PersonaId, 'Post is not linked
 to the Persona.');
        System.assertEquals(existingContact.Id, createdPost.WhoId, 'Post is not linked to
 the Contact');
        System.assertEquals(existingCase.Id, createdPost.ParentId, 'Post is not linked to
 the Case.');
        System.assertEquals(1, [SELECT Id FROM Case].size(), 'There should only be 1
Case.');
        System.assertEquals(false, [SELECT Id, IsClosed FROM Case WHERE Id =
:existingCase.Id].IsClosed, 'Case should be open.');
    }

    static SocialPost getSocialPost(Map<String, Object> socialData) {
        SocialPost post = new SocialPost();
        post.Name = String.valueOf(socialData.get('source'));
        post.Content = String.valueOf(socialData.get('content'));
        post.Posted = Date.valueOf(String.valueOf(socialData.get('postDate')));
        post.PostUrl = String.valueOf(socialData.get('postUrl'));
        post.Provider = String.valueOf(socialData.get('mediaProvider'));
        post.MessageType = String.valueOf(socialData.get('messageType'));
        post.ExternalPostId = String.valueOf(socialData.get('externalPostId'));
        post.R6PostId = String.valueOf(socialData.get('r6PostId'));
        return post;
    }

    static SocialPersona getSocialPersona(Map<String, Object> socialData) {
        SocialPersona persona = new SocialPersona();
        persona.Name = String.valueOf(socialData.get('author'));
        persona.RealName = String.valueOf(socialData.get('realName'));
        persona.Provider = String.valueOf(socialData.get('mediaProvider'));
        persona.MediaProvider = String.valueOf(socialData.get('mediaProvider'));
        persona.ExternalId = String.valueOf(socialData.get('externalUserId'));
        return persona;
    }

    static Map<String, Object> getSampleSocialData(String suffix) {
        Map<String, Object> socialData = new Map<String, Object>();
        socialData.put('r6PostId', 'R6PostId' + suffix);
        socialData.put('r6SourceId', 'R6SourceId' + suffix);
        socialData.put('postTags', null);
        socialData.put('externalPostId', 'ExternalPostId' + suffix);
        socialData.put('content', 'Content' + suffix);
        socialData.put('postDate', '2015-01-12T12:12:12Z');
        socialData.put('mediaType', 'Twitter');
        socialData.put('author', 'Author');
        socialData.put('skipCreateCase', false);
        socialData.put('mediaProvider', 'TWITTER');
        socialData.put('externalUserId', 'ExternalUserId');
```

```
        socialData.put('postUrl', 'PostUrl' + suffix);
        socialData.put('messageType', 'Tweet');
        socialData.put('source', 'Source' + suffix);
        socialData.put('replyToExternalPostId', null);
        socialData.put('realName', 'Real Name');
        return socialData;
    }
}
```

Default Apex Class for Spring ’15 and Summer ’15

```
global virtual class InboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler
 {

    final static Integer CONTENT_MAX_LENGTH = 32000;

    // Reopen case if it has not been closed for more than this number
    global virtual Integer getMaxNumberOfDaysClosedToReopenCase() {
        return 5;
    }

    global virtual String getDefaultAccountId() {
        return null;
    }

    global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
SocialPersona persona, Map<String, Object> rawData) {
        Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
        result.setSuccess(true);
        matchPost(post);
        matchPersona(persona);

        if ((post.Content != null) && (post.Content.length() > CONTENT_MAX_LENGTH)) {
            post.Content = post.Content.abbreviate(CONTENT_MAX_LENGTH);
        }

        if (post.Id != null) {
            handleExistingPost(post, persona);
            return result;
        }

        setReplyTo(post, persona);
        buildPersona(persona);
        Case parentCase = buildParentCase(post, persona, rawData);
        setRelationshipsOnPost(post, persona, parentCase);

        upsert post;

        return result;
    }

    private void handleExistingPost(SocialPost post, SocialPersona persona) {
        update post;
```

```
        if (persona.id != null)
            updatePersona(persona);
    }

    private void setReplyTo(SocialPost post, SocialPersona persona) {
        SocialPost replyTo = findReplyTo(post, persona);
        if(replyTo.id != null) {
            post.replyToId = replyTo.id;
            post.replyTo = replyTo;
        }
    }

    private SocialPersona buildPersona(SocialPersona persona) {
        if (persona.Id == null)
            createPersona(persona);
        else
            updatePersona(persona);
        return persona;
    }


 private void updatePersona(SocialPersona persona) {
  try {
   update persona;
  }catch(Exception e) {
   System.debug('Error updating social persona: ' + e.getMessage());
  }
 }

    private Case buildParentCase(SocialPost post, SocialPersona persona,
    Map<String, Object> rawData){
        Case parentCase = findParentCase(post, persona);
        if (caseShouldBeReopened(parentCase))
            reopenCase(parentCase);
        else if(! hasSkipCreateCaseIndicator(rawData) && (parentCase.id == null ||
parentCase.isClosed))
            parentCase = createCase(post, persona);
        return parentCase;
    }

    private boolean caseShouldBeReopened(Case c){
        return c.id != null && c.isClosed && System.now() <
c.closedDate.addDays(getMaxNumberOfDaysClosedToReopenCase());
    }

    private void setRelationshipsOnPost(SocialPost postToUpdate, SocialPersona persona,
Case parentCase) {
        if (persona.Id != null)
            postToUpdate.PersonaId = persona.Id;
        if(parentCase.id != null)
            postToUpdate.ParentId = parentCase.Id;
    }

    private Case createCase(SocialPost post, SocialPersona persona) {
```

```
        Case newCase = new Case(subject = post.Name);
        if (persona != null && persona.ParentId != null) {
            if (persona.ParentId.getSObjectType() == Contact.sObjectType)
                newCase.ContactId = persona.ParentId;
        }
        if (post != null && post.Provider != null) {
            newCase.Origin = post.Provider;
        }
        insert newCase;
        return newCase;
    }

    private Case findParentCase(SocialPost post, SocialPersona persona) {
        Case parentCase = new Case();
        if (post.ReplyTo != null && (post.ReplyTo.IsOutbound || post.ReplyTo.PersonaId ==
 persona.Id))
            parentCase = findParentCaseFromPostReply(post);
        else if((post.messageType == 'Direct' || post.messageType == 'Private') &&
String.isNotBlank(post.Recipient))
            parentCase = findParentCaseFromRecipient(post, persona);
        return parentCase;
    }

    private Case findParentCaseFromPostReply(SocialPost post){
        List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE Id =
:post.ReplyTo.ParentId LIMIT 1];
        if(!cases.isEmpty())
            return cases[0];
        return new Case();
    }

    private Case findParentCaseFromRecipient(SocialPost post, SocialPersona persona){
        List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE id =
:findReplyToBasedOnRecipientsLastPostToSender(post, persona).parentId LIMIT 1];
        if(!cases.isEmpty())
            return cases[0];
        return new Case();
    }

    private void reopenCase(Case parentCase) {
        SObject[] status = [SELECT MasterLabel FROM CaseStatus WHERE IsClosed = false AND
 IsDefault = true];
        parentCase.Status = ((CaseStatus)status[0]).MasterLabel;
        update parentCase;
    }

    private void matchPost(SocialPost post) {
         if (post.Id != null) return;

     performR6PostIdCheck(post);

        if (post.Id == null){
      performExternalPostIdCheck(post);
     }
```

```
    }


 private void performR6PostIdCheck(SocialPost post){
  if(post.R6PostId == null) return;
  List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE R6PostId = :post.R6PostId
LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
 }


 private void performExternalPostIdCheck(SocialPost post) {
  if (post.provider == 'Facebook' && post.messageType == 'Private') return;
  if (post.provider == null || post.externalPostId == null) return;
  List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE ExternalPostId =
:post.ExternalPostId AND Provider = :post.provider LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
        }
 }


    private SocialPost findReplyTo(SocialPost post, SocialPersona persona) {
        if(post.replyToId != null && post.replyTo == null)
            return findReplyToBasedOnReplyToId(post);
        if(post.responseContextExternalId != null)
            return findReplyToBasedOnExternalPostIdAndProvider(post,
post.responseContextExternalId);
        return new SocialPost();
    }

    private SocialPost findReplyToBasedOnReplyToId(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE id = :post.replyToId LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

   private SocialPost findReplyToBasedOnExternalPostIdAndProvider(SocialPost post, String
 externalPostId){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND ExternalPostId = :externalPostId LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findReplyToBasedOnRecipientsLastPostToSender(SocialPost post,
SocialPersona persona){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE provider = :post.provider AND OutboundSocialAccount.ProviderUserId = :post.Recipient
```

```
 AND ReplyTo.Persona.id = :persona.id ORDER BY CreatedDate DESC LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private void matchPersona(SocialPersona persona) {
        if (persona != null) {
            List<SocialPersona> personaList = new List<SocialPersona>();
            if(persona.Provider != 'Other' && String.isNotBlank(persona.ExternalId)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    Provider = :persona.Provider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
           } else if(persona.Provider == 'Other' && String.isNotBlank(persona.ExternalId)
 && String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    ExternalId = :persona.ExternalId LIMIT 1];
            } else if(persona.Provider == 'Other' && String.isNotBlank(persona.Name) &&
String.isNotBlank(persona.MediaProvider)) {
                personaList = [SELECT Id, ParentId FROM SocialPersona WHERE
                    MediaProvider = :persona.MediaProvider AND
                    Name = :persona.Name LIMIT 1];
            }

            if (!personaList.isEmpty()) {
                persona.Id = personaList[0].Id;
                persona.ParentId = personaList[0].ParentId;
            }
        }
    }

    private void createPersona(SocialPersona persona) {
        if (persona == null || String.isNotBlank(persona.Id) ||
!isThereEnoughInformationToCreatePersona(persona))
            return;

        SObject parent = createPersonaParent(persona);
        persona.ParentId = parent.Id;
        insert persona;
    }

    private boolean isThereEnoughInformationToCreatePersona(SocialPersona persona) {
        return String.isNotBlank(persona.Name) &&
               String.isNotBlank(persona.Provider) &&
               String.isNotBlank(persona.MediaProvider);
    }

    private boolean hasSkipCreateCaseIndicator(Map<String, Object> rawData) {
        Object skipCreateCase = rawData.get('skipCreateCase');
        return skipCreateCase != null &&
'true'.equalsIgnoreCase(String.valueOf(skipCreateCase));
    }
```

```
    global virtual SObject createPersonaParent(SocialPersona persona) {
       String name = persona.Name.trim();
        if (String.isNotBlank(persona.RealName))
            name = persona.RealName.trim();

        String firstName = '';
        String lastName = name;
        if (name.contains(' ')) {
            firstName = name.substringBeforeLast(' ');
            lastName = name.substringAfterLast(' ');
        }

        firstName = firstName.abbreviate(40);
        lastName = lastName.abbreviate(80);

        Contact contact = new Contact(LastName = lastName, FirstName = firstName);
        String defaultAccountId = getDefaultAccountId();
        if (defaultAccountId != null)
            contact.AccountId = defaultAccountId;
        insert contact;
        return contact;
    }

}
```

Default Apex Class for Summer '14 and Winter '14

```
global virtual class InboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler
 {

    // Reopen case if it has not been closed for more than this number
    global virtual Integer getMaxNumberOfDaysClosedToReopenCase() {
        return 5;
    }

    global virtual String getDefaultAccountId() {
        return null;
    }

    global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
SocialPersona persona, Map<String, Object> rawData) {
        Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
        result.setSuccess(true);
        matchPost(post);
        matchPersona(persona);

        if (post.Id != null) {
            handleExistingPost(post, persona);
            return result;
        }

        setReplyTo(post, persona, rawData);
        buildPersona(persona);
```

```
        Case parentCase = buildParentCase(post, persona, rawData);
        setRelationshipsOnPost(post, persona, parentCase);
        upsert post;

        return result;
    }

    private void handleExistingPost(SocialPost post, SocialPersona persona) {
        update post;
        if (persona.id != null)
            update persona;
    }

    private void setReplyTo(SocialPost post, SocialPersona persona, Map<String, Object>
rawData) {
        SocialPost replyTo = findReplyTo(post, persona, rawData);
        if(replyTo.id != null) {
            post.replyToId = replyTo.id;
            post.replyTo = replyTo;
        }
    }

    private SocialPersona buildPersona(SocialPersona persona) {
        if (persona.Id == null)
            createPersona(persona);
        else
            update persona;
        return persona;
    }

   private Case buildParentCase(SocialPost post, SocialPersona persona, Map<String, Object>
 rawData){
        Case parentCase = findParentCase(post, persona);
        if (caseShouldBeReopened(parentCase))
            reopenCase(parentCase);
        else if(! hasSkipCreateCaseIndicator(rawData) && (parentCase.id == null ||
parentCase.isClosed))
            parentCase = createCase(post, persona);
        return parentCase;
    }

    private boolean caseShouldBeReopened(Case c){
        return c.id != null && c.isClosed && System.now() <
c.closedDate.addDays(getMaxNumberOfDaysClosedToReopenCase());
    }

    private void setRelationshipsOnPost(SocialPost postToUpdate, SocialPersona persona,
Case parentCase) {
        if (persona.Id != null)
            postToUpdate.PersonaId = persona.Id;
        if(parentCase.id != null)
            postToUpdate.ParentId = parentCase.Id;
    }
```

```
    private Case createCase(SocialPost post, SocialPersona persona) {
        Case newCase = new Case(subject = post.Name);
        if (persona != null && persona.ParentId != null) {
            if (persona.ParentId.getSObjectType() == Contact.sObjectType)
                newCase.ContactId = persona.ParentId;
        }
        insert newCase;
        return newCase;
    }

    private Case findParentCase(SocialPost post, SocialPersona persona) {
        Case parentCase = new Case();
        if (post.ReplyTo != null && (post.ReplyTo.IsOutbound || post.ReplyTo.PersonaId ==
 persona.Id))
            parentCase = findParentCaseFromPostReply(post);
        else if((post.messageType == 'Direct' || post.messageType == 'Private') &&
post.Recipient != null && String.isNotBlank(post.Recipient))
            parentCase = findParentCaseFromRecipient(post, persona);
        return parentCase;
    }

    private Case findParentCaseFromPostReply(SocialPost post){
        List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE Id =
:post.ReplyTo.ParentId LIMIT 1];
        if(!cases.isEmpty())
            return cases[0];
        return new Case();
    }

    private Case findParentCaseFromRecipient(SocialPost post, SocialPersona persona){
        List<Case> cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE id =
:findReplyToBasedOnRecipientsLastPostToSender(post, persona).parentId LIMIT 1];
        if(!cases.isEmpty())
            return cases[0];
        return new Case();
    }

    private void reopenCase(Case parentCase) {
        SObject[] status = [SELECT MasterLabel FROM CaseStatus WHERE IsClosed = false AND
 IsDefault = true];
        parentCase.Status = ((CaseStatus)status[0]).MasterLabel;
        update parentCase;
    }

    private void matchPost(SocialPost post) {
        if (post.Id != null || post.R6PostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE R6PostId =
:post.R6PostId LIMIT 1];
        if (!postList.isEmpty())
            post.Id = postList[0].Id;
    }

    private SocialPost findReplyTo(SocialPost post, SocialPersona persona, Map<String,
Object> rawData) {
```

```
        if(post.replyToId != null && post.replyTo == null)
            return findReplyToBasedOnReplyToId(post);
        if(rawData.get('replyToExternalPostId') != null &&
String.isNotBlank(String.valueOf(rawData.get('replyToExternalPostId'))))
            return findReplyToBasedOnExternalPostIdAndProvider(post,
String.valueOf(rawData.get('replyToExternalPostId')));
        return new SocialPost();
    }

    private SocialPost findReplyToBasedOnReplyToId(SocialPost post){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE id = :post.replyToId LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

   private SocialPost findReplyToBasedOnExternalPostIdAndProvider(SocialPost post, String
 externalPostId){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE Provider = :post.provider AND ExternalPostId = :externalPostId LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private SocialPost findReplyToBasedOnRecipientsLastPostToSender(SocialPost post,
SocialPersona persona){
       List<SocialPost> posts = [SELECT Id, ParentId, IsOutbound, PersonaId FROM SocialPost
 WHERE provider = :post.provider AND OutboundSocialAccount.ProviderUserId = :post.Recipient
 AND ReplyTo.Persona.id = :persona.id ORDER BY CreatedDate DESC LIMIT 1];
        if(posts.isEmpty())
            return new SocialPost();
        return posts[0];
    }

    private void matchPersona(SocialPersona persona) {
        if (persona != null && persona.ExternalId != null &&
String.isNotBlank(persona.ExternalId)) {
           List<SocialPersona> personaList = [SELECT Id, ParentId FROM SocialPersona WHERE

                Provider = :persona.Provider AND
                ExternalId = :persona.ExternalId LIMIT 1];
            if ( !personaList.isEmpty()) {
                persona.Id = personaList[0].Id;
                persona.ParentId = personaList[0].ParentId;
            }
        }
    }

    private void createPersona(SocialPersona persona) {
        if (persona == null || (persona.Id != null && String.isNotBlank(persona.Id)) ||
!isThereEnoughInformationToCreatePersona(persona))
            return;
```

```
        SObject parent = createPersonaParent(persona);
        persona.ParentId = parent.Id;
        insert persona;
    }

    private boolean isThereEnoughInformationToCreatePersona(SocialPersona persona){
        return persona.ExternalId != null && String.isNotBlank(persona.ExternalId) &&
               persona.Name != null && String.isNotBlank(persona.Name) &&
               persona.Provider != null && String.isNotBlank(persona.Provider) &&
               persona.provider != 'Other';
    }

    private boolean hasSkipCreateCaseIndicator(Map<String, Object> rawData) {
        Object skipCreateCase = rawData.get('skipCreateCase');
        return skipCreateCase != null &&
'true'.equalsIgnoreCase(String.valueOf(skipCreateCase));
    }

    global virtual SObject createPersonaParent(SocialPersona persona) {
        String name = persona.Name;
        if (persona.RealName != null && String.isNotBlank(persona.RealName))
            name = persona.RealName;

        String firstName = '';
        String lastName = 'unknown';
        if (name != null && String.isNotBlank(name)) {
            firstName = name.substringBeforeLast(' ');
            lastName = name.substringAfterLast(' ');
            if (lastName == null || String.isBlank(lastName))
                lastName = firstName;
        }

        Contact contact = new Contact(LastName = lastName, FirstName = firstName);
        String defaultAccountId = getDefaultAccountId();
        if (defaultAccountId != null)
            contact.AccountId = defaultAccountId;
        insert contact;
        return contact;
    }
}
```

Default Apex Class for Winter ’13 and Spring ’14

```
global virtual class InboundSocialPostHandlerImpl implements Social.InboundSocialPostHandler
 {

    // Reopen case if it has not been closed for more than this number
    global virtual Integer getMaxNumberOfDaysClosedToReopenCase() {
        return 5;
    }

    global virtual Boolean usePersonAccount() {
```

```
        return false;
    }

    global virtual String getDefaultAccountId() {
        return null;
    }

    global Social.InboundSocialPostResult handleInboundSocialPost(SocialPost post,
SocialPersona persona, Map<String, Object> rawData) {
        Social.InboundSocialPostResult result = new Social.InboundSocialPostResult();
        result.setSuccess(true);
        matchPost(post);
        matchPersona(persona);

        if (post.Id != null) {
            update post;
            if (persona.id != null) {
                update persona;
            }
            return result;
        }

        findReplyTo(post, rawData);

        Case parentCase = null;
        if (persona.Id == null) {
            createPersona(persona);
            post.PersonaId = persona.Id;
        }
        else {
            update persona;
            post.PersonaId = persona.Id;
            parentCase = findParentCase(post, persona, rawData);
        }

        if (parentCase == null) {
            parentCase = createCase(post, persona);
        }

        post.ParentId = parentCase.Id;

        insert post;

        return result;
    }

    private Case createCase(SocialPost post, SocialPersona persona) {
        Case newCase = new Case(
            subject = post.Name
        );
        if (persona != null && persona.ParentId != null) {
            if (persona.ParentId.getSObjectType() == Contact.sObjectType) {
                newCase.ContactId = persona.ParentId;
            }
```

```
            else if (persona.ParentId.getSObjectType() == Account.sObjectType) {
                newCase.AccountId = persona.ParentId;
            }
        }
        insert newCase;
        return newCase;
    }

   private Case findParentCase(SocialPost post, SocialPersona persona, Map<String, Object>
 rawData) {
        SocialPost replyToPost = null;
        if (post.ReplyTo != null && (post.ReplyTo.IsOutbound || post.ReplyTo.PersonaId ==
 persona.Id)) {
            replyToPost = post.ReplyTo;
        }
        else if (post.MessageType == 'Direct' && String.isNotBlank(post.Recipient)) {
            // find the latest outbound post that the DM is responding to
            List<SocialPost> posts = [SELECT Id, ParentId FROM SocialPost WHERE
OutboundSocialAccount.ProviderUserId = :post.Recipient AND ReplyTo.Persona.Id = :persona.Id
 ORDER BY CreatedDate DESC LIMIT 1];
            if (!posts.isEmpty()) {
                replyToPost = posts[0];
            }
        }

        if (replyToPost != null) {
            List<Case>  cases = [SELECT Id, IsClosed, Status, ClosedDate FROM Case WHERE
Id = :replyToPost.ParentId];
            if (!cases.isEmpty()) {
                if (!cases[0].IsClosed) return cases[0];
                if (cases[0].ClosedDate >
System.now().addDays(-getMaxNumberOfDaysClosedToReopenCase())) {
                    reopenCase(cases[0]);
                    return cases[0];
                }
            }
        }

        return null;
    }

    private void reopenCase(Case parentCase) {
        SObject[] status = [SELECT MasterLabel FROM CaseStatus WHERE IsClosed = false AND
 IsDefault = true];
        parentCase.Status = ((CaseStatus)status[0]).MasterLabel;
        update parentCase;
    }

    private void matchPost(SocialPost post) {
        if (post.Id != null || post.R6PostId == null) return;
        List<SocialPost> postList = [SELECT Id FROM SocialPost WHERE R6PostId =
:post.R6PostId LIMIT 1];
        if (!postList.isEmpty()) {
            post.Id = postList[0].Id;
```

```
        }
    }

    private void findReplyTo(SocialPost post, Map<String, Object> rawData) {
        String replyToId = (String)rawData.get('replyToExternalPostId');
        if (String.isBlank(replyToId)) return;
        List<SocialPost> postList = [SELECT Id, ParentId, IsOutbound, PersonaId FROM
SocialPost WHERE ExternalPostId = :replyToId LIMIT 1];
        if (!postList.isEmpty()) {
            post.ReplyToId = postList[0].id;
            post.ReplyTo = postList[0];
        }
    }

    private void matchPersona(SocialPersona persona) {
        if (persona != null && String.isNotBlank(persona.ExternalId)) {
           List<SocialPersona> personaList = [SELECT Id, ParentId FROM SocialPersona WHERE

                ((Provider != 'Other' AND Provider = :persona.Provider) OR
                 (Provider = 'Other' AND MediaProvider != null AND MediaProvider =
:persona.MediaProvider)) AND
                ((ExternalId != null AND ExternalId = :persona.ExternalId) OR
                 (ExternalId = null AND Name = :persona.Name)) LIMIT 1];
            if ( !personaList.isEmpty()) {
                persona.Id = personaList[0].Id;
                persona.ParentId = personaList[0].ParentId;
            }
        }
    }

    private void createPersona(SocialPersona persona) {
        if (persona == null || persona.Id != null || String.isBlank(persona.ExternalId)
|| String.isBlank(persona.Name) ||
                String.isBlank(persona.Provider)) return;

        if (isPersonaAccountEnabled()){
            Account account = createPersonAccount(persona);
            persona.ParentId = account.Id;
        }
        else {
            Contact contact = createContact(persona);
            persona.ParentId = contact.Id;
        }
        insert persona;
    }


    private Boolean isPersonaAccountEnabled() {
        if (!usePersonAccount()) return false;
        Map<String, Object> accountFields = Schema.SObjectType.Account.fields.getMap();
        return accountFields.containsKey('IsPersonAccount');
    }

    private Account createPersonAccount(SocialPersona persona) {
```

```
        Account account = new Account(
            Name = persona.Name
        );
        insert account;
        return account;
    }

    private Contact createContact(SocialPersona persona) {
        String name = persona.RealName;
        if (String.isBlank(name)) {
            name = persona.Name;
        }

        String firstName = '';
        String lastName = 'unknown';
        if (!String.isBlank(name)) {
            firstName = name.substringBeforeLast(' ');
            lastName = name.substringAfterLast(' ');
            if (String.isBlank(lastName)) {
                lastName = firstName;
            }
        }

        Contact contact = new Contact(
            LastName = lastName,
            FirstName = firstName
        );
        String defaultAccountId = getDefaultAccountId();
        if (defaultAccountId != null) {
            contact.AccountId = defaultAccountId;
        }
        insert contact;
        return contact;
    }
}
```

Social Action Tips

Use the social action on the case or lead feed to respond to social posts. Choose an appropriate message type when composing a response. For example, you can choose to reply with a direct message on Twitter rather than a public tweet.

We recommend that the layout of the social action includes the following fields, at minimum.

Field	Description
In Reply To	The social post you are replying to and its content. Use the Reply, Retweet, and Comment links in the feed to add content to a specific item in the feed.
Managed Social Account	You must have access to the managed social account by a profile or permission set. Use the drop-down to change to another account you have access to.
Message Type	By default the message type is set to Reply for inbound posts. Use the drop-down to change to another valid message type.
Content	All outbound content must be unique for the interaction, you can't send the same content in the same conversation. All Twitter replies must start with a handle: @[social handle].

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Social Customer Service is available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To send and receive social media posts or messages:

- "Social Objects"
  AND
  "Social Publisher"
  AND
  Case Feed enabled
  AND
  Social account

Note: The Social Customer Service setup pages, the moderation and authorization pages, and the Reply action in the case feed are only available in Salesforce Classic.

If your posts require approval before they are sent, you can click **Submit for Approval** to start the review process. You can **Recall** it before it is approved or rejected. If a post is rejected, you can **Retry** a rewritten post. When your post is approved, it is automatically published.

Here are some tips for working with social networks.

- You can like, unlike, view source, and delete social media from the case feed while in Lightning Experience. However, you must reply to posts in Salesforce Classic.
- URLs in a social post are turned into clickable links.
- When deleting posts, consider that Twitter Direct Messages behave like to emails. For example, the sender can delete their direct message from a conversation view. However, receiver has that direct message in their conversation view until they choose to delete it.
- Speaking of Twitter Direct Messages, Twitter has a preference to "Receive Direct Messages from Anyone". Therefore, depending on if this permission is set on the recipient's or your account, you may not have to follow each other to direct message.
- If your Twitter settings allow you to receive direct messages from anyone, you can send deep links to invite users to direct message conversations. To send a deep link direct message invitation, paste this link into your outbound message:
  `https://twitter.com/messages/compose?recipient_id={your Twitter account's numeric user ID}`

You can find your twitter account's numeric ID on twitter.com by going to **Your Twitter User** > **Settings** > **Your Twitter Data**. Twitter handles the URLs and the rendering in their native clients.

- Agents can use the **View Source** link to go to the original post within its social network.
- In Salesforce1, agents can see and reply to social content from mobile devices.
- Only change the Status picklist field on social posts if you are working with outbound posts. If an agent manually sets the status on the inbound social post detail page, the social posts in the case feed may not match. We recommend removing the Status field on the inbound social post detail page layout. For example, if you change the Status of an inbound post to Sending, the Reply link in the case feed item disappears until you change the status back to None.

For Twitter accounts, agents can use case and lead feeds to see the content that they are responding to, retweet, mark as Like and follow tweets, send replies to tweets and direct messages, and delete tweets managed by your social accounts.

For Facebook accounts, cases and leads are created from your managed Facebook page and agents can use the feeds to see the content that they are replying to, like posts and comments, send posts, comments, replies, and private messages, and delete posts managed by your social accounts.

Here are tips for dealing with some possible error messages.

Message	Action
You can't send a direct message to this Twitter user because the user is not following you.	Use a reply to ask the Twitter user to follow your managed social account. Once they are following you, send them a direct message.
Whoops! You already said that... Change your message and try again.	You can't post the same text twice. Change your content and send again.
Your content is too long.	Reduce your content to 140 characters or less. For Twitter replies, the handle is included in the character count.
Twitter replies must begin with a handle.	The content field for a Twitter reply must be in the form: @[sender's handle] message text. Ensure that there is a space between the sender's handle and your message.
Your response message type must be compatible with the original post's message type.	Change the message type to match the original message.
Your login to Social Studio has failed. The username or password may be incorrect. Update your credentials or reset your password.	An administrator must edit the Social Studio credentials on the Social Media settings page. Note: When an administrator makes a copy of or refreshes a Sandbox organization, a new organization is created, with a new ID, making the Social Studio login invalid.
Your post did not send.	We recommend creating a workflow to notify the case owner that an attempt to send a response via Twitter has failed.

SEE ALSO:

Implementing Social Customer Service

Manage Social Posts

A social post is a Salesforce object that represents a post on a social network such as Facebook or Twitter.

Note: The Social Customer Service setup pages, the moderation and authorization pages, and the Reply action in the case feed are only available in Salesforce Classic.

The Social Posts tab or object is a collection of information about a post from a person or company on a social network, such as Twitter or Facebook. The available information for a post varies depending on the social network. You can view and manage social posts.

Note: For inbound posts, setting a Status picklist value on the social post detail page does not stay with the post, as this field is for outbound posts only.

1. Click the **Social Posts** tab.
2. Optionally, select a view.

   The list defaults to those recently viewed. Select a `View` or create one to filter the list of posts. If your organization has moderation enabled, select **Social posts without case** to view and either create a case for or ignore posts. You can also create a view to fit your needs.
3. Click the social post name you'd like to manage or click **New Social Post** to create a post.

   If you selected a view, you can click **Edit** or **Del** (delete) as appropriate.

   Note: On the Social Post tab, you can only create, edit, and delete posts in your Salesforce organization, not on the social networks.
4. To manage posts without cases, select the posts you'd like to either create a case for or ignore and click **Create Case** and **Ignore** as appropriate.

   For example, an agent can ignore a Facebook post of "I love you guys!" as it does not warrant a case.

   If you are using the Social Customer Service Starter Pack, you can enable case moderation on the Social Accounts tab in Setup, see Set Up Social Customer Service on page 4. To enable moderation through Social Studio see, Enable Moderation for Social Customer Service on page 8.
5. If you have Approvals enabled, Social Posts tab has a `Social posts pending approval` list view that allows you to review multiple pending posts and approve or reject them as desired.

   Note: Once approvals are enabled, the Approve Posts and Reject Posts buttons remain on the Social Posts tab. However they don't work for inbound and posts not needing approval.

   Tip: If you approve a post from the Social posts pending approval list view and a system interruption, session timeout, or other unexpected issue prevents the post from being published on the intended social network, an error message displays on the individual case only, not on the list view. To help honor any commitments, your company may have regarding response times on social networks, after approving posts from the list view, we recommend checking the posts' statuses to ensure that they were sent successfully and don't need to be resent.

On the social post detail page you can:

- View, edit, and create the post's content and information.

  Note: The information varies depending on the social network the persona is from.

  Don't forget to click **Save** to save changes or create a post.

EDITIONS

Available in: Lightning Experience and Salesforce Classic

Social personas and posts are available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To install and deploy Social Studio for Salesforce:
- "Customize Application"

- If your organization has moderation enabled, you can create a case for a post or ignore it if it does not warrant a case.
- Delete the post in your Salesforce organization.

  Note: Social posts are not deleted when their parent record, usually a case, is deleted. Similarly, if a social post is associated with an account, contact, or lead through the polymorphic Who field, deleting any of those related records does not affect the social post.

You can reply to a social post from the case or lead feeds only, not the Social Posts tab.

Manage Social Personas

A social persona is a Salesforce object that represents a contact's profile on a social network such as Facebook, or Twitter.

Note: The Social Customer Service setup pages, the moderation and authorization pages, and the Reply action in the case feed are only available in Salesforce Classic.

The Social Personas tab or object is a collection of publicly available information about a person or company from Twitter or Facebook. A Persona is relative to the social network and there can be multiple personas attached to a single contact. You can edit or delete a persona but you can't manually create a social persona from Salesforce. The personas are created from public information on social networking sites. You can view and manage your social persona records like other records in Salesforce.

Note: Social persona fields many have maximum character lengths set by standard or custom Salesforce limits. For example, the first name field is limited to 40 characters. If a social persona with a first name longer than 40 characters is created from an inbound social post, the first name is truncated at the 40th character.

EDITIONS

Available in: Lightning Experience and Salesforce Classic

Social personas and posts are available in API enabled **Professional** editions, **Enterprise**, **Performance**, and **Unlimited** editions.

USER PERMISSIONS

To install and deploy Social Studio for Salesforce:
- "Customize Application"

1. Click the **Social Personas** tab.
2. Optionally, select a view.

   The list defaults to those recently viewed. Select **All** in the `View` drop-down to show all social personas in your organization. You can also create a view to fit your needs.
3. Click the social handle you'd like to manage.

   If you selected a view, click **Edit** or **Del** (delete) as appropriate.

   Warning: If you delete a social persona through the Social Accounts and Contacts feature, all related social posts are also deleted.

On the social persona detail page you can:

- View and edit the contact's available information for that social network.

  Note: The information varies depending on the social network the persona is from.
- Delete the social persona from your organization.
- Create, edit, and delete social posts.
- View which social network created the persona, in the `Source App` field. This field is set on creation and is not updateable. Social Personas created prior to the Summer '15 release do not have this field.

Warning: There is no field level security and you can't control who can create, read, edit, or delete Social Personas. Anyone in your organization can see all the data on a Social Persona object.