Sitecore Social Connected 8.0

All the official Sitecore documentation.

sitecore®
Own the experience™

Logging in to a Sitecore website using social network credentials

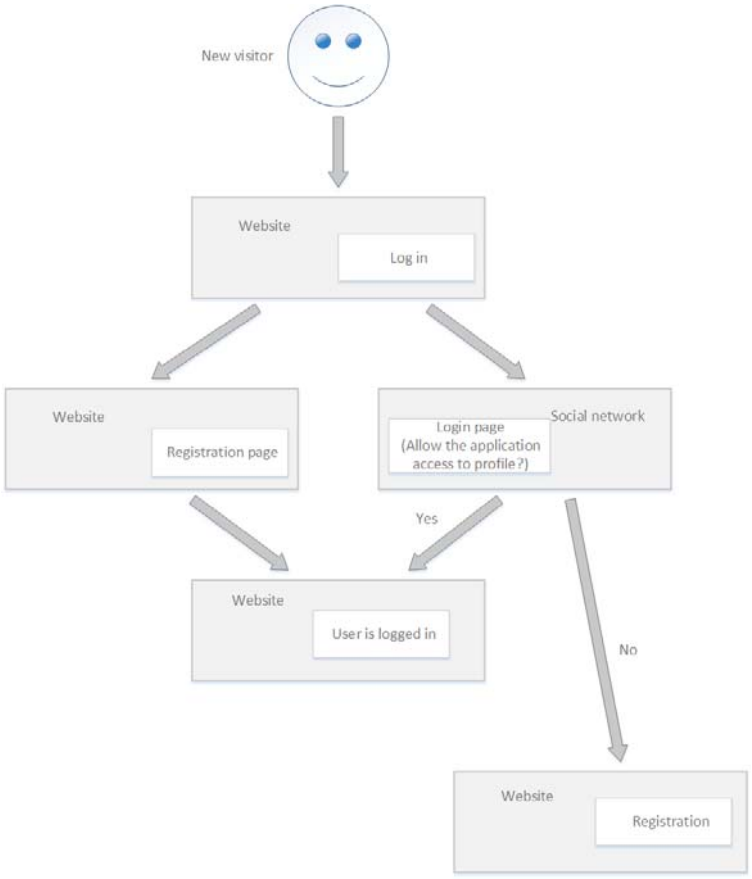
The Social Connected module lets website visitors log in to the website with their social network credentials. This feature is called Social Connector. Sitecore does not access the visitor's credentials; it uses the API to communicate with the social network. If a visitor does not grant access to their profile information, the system redirects them back to the website. The visitor still has to log in to the website.

When a website visitor logs in using their social network credentials, the Social Connected module receives all the information from the visitor's profile that they have allowed the social network to share.

The module saves the social profile information to a Sitecore user profile in the extranet domain and to a contact profile. A user profile stores service information, such as access tokens, application ID, and so on. A contact profile stores the rest of the information, such as interests, sex, age, and so on.

Website visitors can use Social Connector to log in to the website with their social network credentials, thereby avoiding the registration process. Social Connector also enables the website to collect more information about a website visitor from their social network profile.

The process of logging in to a Sitecore website



Send feedback about the documentation to docsite@sitecore.net.

Configure advanced logging

The Social Connected module logs service information in the log files. You can enable more detailed logging. You can also customize logging to log information about custom objects and classes.

Enable detailed logging

To enable the registering of the tracing information by Social Connected in the log files, in the `website\app_config\include\social\sitecore.social.config` file, set the `Social.Logging.TraceToLog` setting to *true*.

Customize logging

To log information about your custom objects and static classes using the Social Connected API, you must implement formatters.

An object type or a static class must have an appropriate formatter that knows how to log this particular type.

Formatters are classes that implement the `Sitecore.Social.Infrastructure.Logging.IEntityFormatter` interface. Formatters are decorated with the `Sitecore.Social.Infrastructure.Logging.FormatterForAttribute` attribute that specifies the type to log.

If there is no formatter defined for the object type and the type is marked with the `System.SerializableAttribute` attribute, the system uses the default formatter that uses the SoapFormatter class to serialize objects into the SOAP format.

When you have implemented the formatters, you must specify the assemblies in which the formatters are stored:

1. Open the `website\app_config\include\social\sitecore.social.config` file.
2. In the `entityFormatterAssemblies` node, in the `assembly name` parameter, specify the .dll files in which the module searches formatters:

```
<logging>
        <!-- Assemblies, used to search entity formatters for logging. -->
        <entityFormatterAssemblies>
          <assembly name="Sitecore.Social.Infrastructure.Logging.dll" />
        </entityFormatterAssemblies>
      </logging>
```

The `LogObject`, `LogStatic` and `LogEnumerable` methods of the Social Connected log manager (the `Sitecore.Social.Infrastructure.Logging.LogManager` class) now use your formatters for the appropriate types.

For a detailed description of the Social Connected logging API with examples, see the `Sitecore.Social.Infrastructure.Logging` namespace in the Social Connected API document.

Configure posting interval

A message must meet all of the following requirements to be automatically posted to a social network:

- In the message configuration window, the Post automatically check box is selected.
- The message is either in the final state of a workflow or not in a workflow.
- The message has never been posted before.
- The corresponding content item is published after the message was created.

The Social Connected module uses the `Sitecore.Social.Client.Tasks.PostFlaggedMessages` scheduling agent to post content messages automatically after you publish the Sitecore items that have content messages assigned to them.

This scheduling agent is ready to use, and it is configured to post messages that are ready to be posted to social networks every hour. Therefore, if you publish a Sitecore item, the corresponding message is posted next time the scheduling agent runs. As a result, there is a delay between when an item is published and when the corresponding message is posted.

To change the posting interval:

1. Open the `[website_root]\app_config\include\social\sitecore.social.config` file.
2. In the `scheduling` section, set the `interval` value:

```
<!-- The agent that posts messages that are marked to be posted automatically. -->
      <agent type="Sitecore.Social.Client.Tasks.PostFlaggedMessages, Sitecore.Social.Client" method="Run" interval="01:00:00">
        <param desc="Social accounts root (item path or ID)">/sitecore/system/social/accounts</param>
      </agent>
```


The message search

The Social Connected module stores messages in the *Sitecore/Social/Messages* folder. This folder is an item bucket that has a configured index that the Social Connected module uses to search for the messages. This index is based on the Lucene or Solr provider, and the `Sitecore.ContentSearch` API.

The following table shows the files in which you can change the index configuration for Lucene. The configuration files for Solr are identical but contain Solr in the name of the files instead of Lucene.

Configuration file	Section
`\Website\app_config\Include\social\Sitecore.Social.Lucene.IndexConfiguration.config`	`social/indexConfigurations`
`\Website\app_config\Include\social\Sitecore.Social.Lucene.Index.Master.config`	`sitecore/contentSearch/configuration/indexes/index`
`\Website\App_Config\Include\social\Sitecore.Social.Lucene.Index.Web.config`	`sitecore/contentSearch/configuration/indexes/index`

`\Website\App_Config\Include\social\Sitecore.social.config`	`sitecore/settings`

Note

The cache helps you to get the latest indexed data in a container. Other API methods, for example, `GetMessagesByWorkflowState` or `GetMessagesByAccount`, depend on the corresponding job execution.

Solr configuration

Social Connected uses index names as names for Solr cores: `social_messages_master` and `social_messages_web` for the *master* and *web* databases respectively. If you give Social Connected cores different names, you must specify these names in the `Sitecore.Social.Solr.Index.Master.config` and `Sitecore.Social.Solr.Index.Web.config` files, in the `<param desc="core"></param>` element.

Configuration for scaled environment

If you install the Social Connected module package on the Content Delivery server, you must edit the `\Website\App_Config\Include\social\Sitecore.Social.Lucene.Index.Web.config` or `\Website\App_Config\Include\social\Sitecore.Social.Solr.Index.Web.config` file specifying the name of the Content Delivery server database in the database section. If you use a single common server for the Content Management and Content Delivery servers with the Master and Web databases, you can use the default settings.

If you create a custom index for the Content Management or Content Delivery server, you must specify the index ID in the corresponding settings:

- `Social.Messages.SearchIndex.Master` – the name of the Content Management server index
- `Social.Messages.SearchIndex.Web` – the name of the Content Delivery server index

The Social Connected pipelines

The Social Connected module uses six pipelines. The pipelines are defined in the `Sitecore.Social.config` file. You can extend and customize the pipelines by adding custom processors or replacing the default ones.

The social.createMessage pipeline

The `social.createMessage` pipeline is used when the module creates a message for a social network. It includes the following processors:

Processor name	Description
`GetRoot`	Gets a path in the content tree for the message item
`CreateItem`	Creates two items under the messages root item: • Message • Posting configuration
`SaveData`	Saves message data from its fields

The social.readMessage pipeline

The `social.readMessage` pipeline reads all the messages for the current item from the content tree and returns the messages collection. It includes one processor:

Processor name	Description
`ReadMessages`	Reads all the messages for the specified message IDs

The social.postMessage pipeline

The `social.postMessage` pipeline posts the message to the social network. It includes the following processors:

Processor name	Description
`ResolveMessagePostingProvider`	Selects a posting provider to post the message
`PostMessage`	Posts the message to the social network

The social.buildMessage pipeline

The `social.buildMessage` pipeline builds the message before it is posted to the social network. It returns the message object that is ready to be posted. It includes the following processors:

Processor name	Description
`ResolveRenderer`	Returns the message builder for the posting configuration (goal or content)
`BuildMessage`	Builds the message
`ReplaceTokens`	Replaces tokens with their values

The social.matchUser pipeline

The`social.matchUser`pipeline uses the data provided by a user to search for an existing user. If it cannot find the existing user, and the `CreateNewUserIfNoExistingFound` property of the pipeline is set to *true*, it creates a new user.

Processor name	Description
`PrepareUserData`	Creates a username and email for the user based on the user basic data.
`FindByEmail`	Uses the email information provided by the user to search for an existing user
`FindByDomain`	Searches for an existing user by domain
`FindByNetworkCredentials`	Uses the network credentials (ID) to search for an existing user
`FindByFacebookIdsForBusiness`	This processor is specific to the Facebook network. It searches for an existing user by application-scoped IDs, if you use the Business Manager. Note This Social Connected functionality is available starting from Sitecore XP 8 Update-2.
`CreateUser`	Creates a new user

The social.initializeApi pipeline

The `social.initializeApi` pipeline initializes social API. It includes one processor:

Processor name	Description
`IoCInitialization`	Initializes the inversion of control (IoC) container

Use the Social Connected configuration framework

The configuration framework in Social Connected lets you map the Sitecore configuration to C# classes. Using the configuration framework, you can encapsulate the source of the configuration settings, removing dependency from Sitecore. This lets you easily substitute configuration in unit tests.

To use the Social Connected configuration framework to map the Sitecore configuration to C# classes:

1. Create a settings class that represents a specific configuration using `Sitecore.Social.Configuration.Model.ConfigurationData` as a base class.

   Note

   The class is located in the `Sitecore.Social.Configuration.dll` assembly.

Use the following code as a sample:

```
namespace Sitecore.Social.Tumblr.Configuration
{
  using System;
  using Sitecore.Social.Configuration.Managers;
  using Sitecore.Social.Configuration.Model;
  /// <summary>
  /// Represents Tumblr specific settings.
  /// </summary>
  [Serializable]
  public class SettingsConfiguration : ConfigurationData
  {
    /// <summary>
    /// A read-only instance of the <see cref="SettingsConfiguration"/> class whose value represents a not defined configuration s
    /// </summary>
    public static readonly SettingsConfiguration Empty = new SettingsConfiguration();
    /// <summary>
    /// Gets or sets the default share goal.
    /// </summary>
    /// <value>
    /// The default share goal.
    /// </value>
    public string DefaultShareGoal { get; set; }
    /// <summary>
    /// Gets a value indicating whether the configuration is empty.
    /// </summary>
    /// <value>
    ///   <c>true</c> if configuration is empty; otherwise, <c>false</c>.
    /// </value>
    public override bool IsEmpty
    {
      get
      {
        return this == Empty;
      }
    }
  }
}
```

2. Implement the Configuration Manager: create a class that gets the configuration settings from Sitecore. Use the following sample code:

```
namespace Sitecore.Social.Tumblr.Configuration
{
  using Sitecore.Configuration;
  using Sitecore.Social.Configuration.Managers;
  using Sitecore.Social.Configuration.Model;
  /// <summary>
  /// Represents settings configuration manager for Tumblr.
  /// </summary>
  public class SettingsConfigurationManager : ConfigurationManager
  {
    /// <summary>
    /// Gets the configuration data.
    /// </summary>
    /// <returns></returns>
    public override ConfigurationData GetConfigurationData()
    {
      return new SettingsConfiguration
               {
```

```
                DefaultShareGoal = Settings.GetSetting("Social.Tumblr.DefaultShareGoal", "Tumblr Share")
            };
        }
    }
}
```

3. Map the Configuration Manager to the settings class by adding the `Sitecore.Social.Configuration.Managers.ConfigurationManager` attribute to the `SettingsConfiguration` class described in step 1, and set the `ManagerType` parameter to the type of the Configuration Manager that you implemented in step 2. For example:

```
/// <summary>
/// Represents Tumblr specific settings.
/// </summary>
[Serializable]
[ConfigurationManager(ManagerType = typeof(SettingsConfigurationManager))]
public class SettingsConfiguration : ConfigurationData
{
```

4. Use the `ConfigurationFactory` class to get the appropriate settings class instance. The following code contains an example of how to retrieve the value of the `DefaultShareGoal` setting using the Social Connected configuration framework:

```
var tumblrSettingsConfiguration = ExecutingContext.Current.IoC.Get<IConfigurationFactory>().Get<SettingsConfiguration>();
var script = string.Format(
        CultureInfo.CurrentCulture,
        @"<script>document.getElementById('{0}').addEventListener('click', function(){{{1}}})</script>",
        this.tumblrShareButton.ClientID,
        SharingHelper.GetShareCallbackScript(this.GoalName, tumblrSettingsConfiguration.DefaultShareGoal));
```

You must add references to the following assemblies to make the code above work:

- `Sitecore.Social.Configuration.dll`
- `Sitecore.Social.Infrastructure.dll`
- `Ninject.dll`

Note

Social Connected uses Ninject to resolve dependencies.

Walkthrough: Configure Social Connected for Klout Scores

The Social Connected module can access information about a contact's Klout Score if the contact logs in to your website with Twitter or Google+ credentials, or the contact attaches one or both of these social networks to their profile. The Social Connected module transfers the Klout Score to the Sitecore analytics database – Mongo DB.

You can use the Klout Score to personalize your website.

Important

When you register your app on Klout's website, you enter into an agreement with Klout, who is an independent third party. Sitecore is not responsible for Klout's services or any losses arising from usage of these services.

This walkthrough describes how to:

- Configure the Social Connected module for Klout Scores
- Create a Klout application
- Create a Sitecore item for the Klout application
- Configure the Twitter or Google+ log-in control

Configure the Social Connected module for Klout Scores

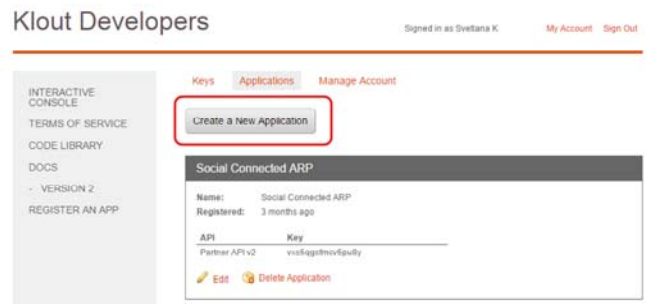
Before you can access a contact's Klout Score, you need to ensure that Klout is enabled:

- In the `Website\app_config\include\social` folder, rename Sitecore.Social.Klout.config.disabled to Sitecore.Social.Klout.config.

Create a Klout application

The Social Connected module receives visitors' Klout Scores from Klout via the application. To create a Klout application:

1. Go to https://secure.mashery.com/login/developer.klout.com/ and click Register to build your awesome app.
2. On the Applications tab, click Create a New Application.



3. Fill in the required fields.

Create a Sitecore item for the Klout application

After you have created a Klout application, you must create a Sitecore item for it. This Sitecore item contains connection keys that Sitecore uses to get the information from Klout.

To create a Sitecore item for the Klout application:

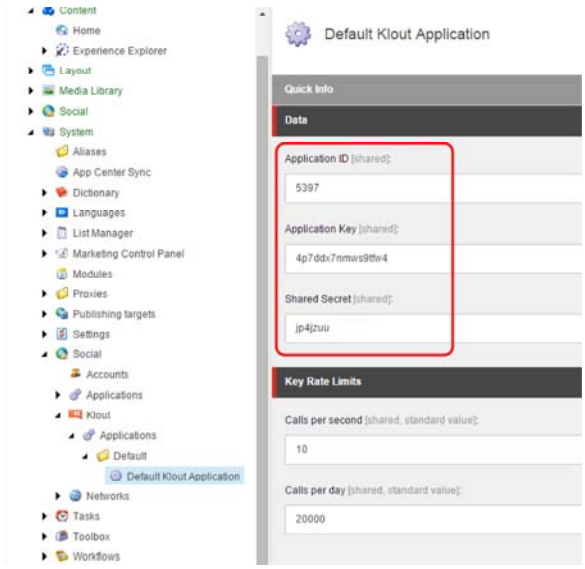
1. In Sitecore, in the Content Editor, open the *sitecore/system/social/Klout/applications/default* folder.
2. Create an item based on the */sitecore/templates/System/Social/Klout/Klout Application* template.

   Important

   The *sitecore/system/social/Klout/applications/default* folder must contain only one Klout item. If you want to use several applications for Klout, you must create a new item for each one in the *sitecore/system/social/Klout/applications* folder.

3. In the new item, fill in the following fields:

Field	Value
Application ID	The ID value of the Klout application. To get the ID, go to http://developer.klout.com/apps/myapps and then click Edit application. You can see the ID in the last part of the URL: http://developer.klout.com/apps/edit/ID.
Application Key	The Key value of the Klout application. You can get the key on this page: http://developer.klout.com/apps/mykeys.
Shared Secret	The Shared Secret value of the Klout application. You can get this value on this page: http://developer.klout.com/apps/mykeys



Configure the Twitter or Google+ log-in control

Klout does not have its own Sitecore log-in control. Twitter and Google+ log-in controls use the Klout application to get the Klout Score. By default, the Twitter and Google+ log-in controls are configured to use the Klout application from the `/sitecore/system/Social/Klout/Applications/Default` folder.

If you want to set up a control to use an application that is not included in the `Default` folder:

1. In Sitecore, in the Content Editor, click the item that contains either the *Login with Twitter* or the *Login with Google* login control.
2. On the ribbon, on the Presentation tab, click Details.
3. In the Layout details dialog box, click either the *Login with Twitter* or the *Login with Google* control.
4. In the Klout Application field, select the application that you want the control to use.

Send feedback about the documentation to docsite@sitecore.net.

Walkthrough: Configuring Share buttons using Sitecore MVC

You can configure and place Share buttons from different social networks on your website. When a website visitor clicks a Share button, an appropriate message with the link to the website is displayed in the social network. You can use either Sitecore sublayouts or Sitecore MVC renderings to create Share buttons, depending on the functionality you use on your website. In this walkthrough, Tumblr is used as an example.

Follow this walkthrough if you use Sitecore MVC.

This walkthrough describes how to:

Create a Share button campaign

You can assign any campaign to a Share button. This feature uses Sitecore xDB. You can use standard Sitecore campaigns or create new ones for Social Connected.

To create a campaign for a Share button:

1. In the Content Editor, navigate to `/sitecore/system/Marketing Control Panel/Campaigns/Social` and click the relevant social network campaign group, for example, *Tumblr*.
2. Under the *Tumblr* category item, create a new item based on the *Campaign* template (`/sitecore/templates/System/Analytics/Campaign`). Name the new item, for example, *Tumblr Share Buttons*.

Note

Sitecore xDB requires a campaign to be deployed before you can use it. To deploy the new campaign, on the Home tab, click Edit and then click Deploy.

Create a Share button goal

Some social networks allow users to subscribe to Share button events. For example, Twitter's Tweet button has a 'tweet' event, which occurs when a user publishes a tweet using the Tweet button.

You can use standard Sitecore goals or create new ones for Social Connected. To create a goal for a Share button:

1. In the Content Editor, navigate to `/sitecore/system/Marketing Control Panel /Goals`.
2. In the *Goals* folder, create a new goal. Name it, for example, *Tumblr Share*.

Note

Sitecore xDB requires a goal to be deployed before you can use it. To deploy the new goal, on the **Home** tab, click **Edit** and then click **Deploy.**

Create a Share button MVC controller

Social Connected uses the Sitecore controller rendering feature.

To create a Share button MVC controller:

1. Create a controller in the *Sitecore.Social.Tumblr.Client.Mvc* project.
2. Use the `Sitecore.Social.Client.Mvc.Areas.Social.Controllers.SharingController` as a base class:

```
namespace Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Controllers
{
  using System.Web.Mvc;
  using Sitecore.Globalization;
  using Sitecore.Social.Client.Common.Helpers;
  using Sitecore.Social.Client.Mvc.Areas.Social.Controllers;
  using Sitecore.Social.Client.Mvc.Areas.Social.Models;
  /// <summary>
  /// Tumblr sharing button controller.
  /// </summary>
  public class TumblrSharingController : SharingController
  {
    /// <summary>
    /// Gets Tweet button view.
    /// </summary>
    /// <returns>Tweet button view.</returns>
    public PartialViewResult TumblrButton()
    {
```

```
          return this.PartialView(
            "~/Areas/Social/Views/Sharing/TumblrButton.cshtml",
            new ShareButtonViewModel
            {
              SharePageUrlWithAnalyticsParameters = SharingHelper.GetSharePageUrlWithAnalyticsParameters(this.CampaignId),
              Text = Translate.Text(Common.Texts.ShareOnTumblr)
            });
        }
      }
    }
```

Create a Share button MVC view

To create a view for the MVC Share button:

1. Use the relevant code to create a view for the MVC share button. For example, for Tumblr you can find the original markup of the Tumblr Share button at https://www.tumblr.com/buttons.
2. Store this view with the other views, for example, at: `/Areas/Social/Views/Sharing/TumblrButton.cshtml`

Sample code

```
@model Sitecore.Social.Client.Mvc.Areas.Social.Models.ShareButtonViewModel
<div style="float: right;">
  <a href="http://www.tumblr.com/share/link?url=@Url.Encode(Model.SharePageUrlWithAnalyticsParameters)"
     title="@Model.Text"
     style="display: inline-block; text-indent: -9999px; overflow: hidden; width: 81px; height: 20px; background: url('https://platforı
    @Model.Text
  </a>
  <script type="text/javascript">
    (function (scriptTagId) {
      var tumblrScript = document.getElementById(scriptTagId);
      if (!tumblrScript) {
        tumblrScript = document.createElement("script");
        tumblrScript.src = "http://platform.tumblr.com/v1/share.js";
        tumblrScript.id = scriptTagId;
        document.body.appendChild(tumblrScript);
      }
    })('tumblrShareScript');
  </script>
</div>
```

Create the Share button MVC controller rendering

To create a connector MVC controller rendering:

1. Navigate to `/sitecore/layout/Renderings/Social MVC/Sharing/` and in the *Sharing* item, create a new MVC controller rendering. For convenience, you can duplicate an existing one.
2. In the Controller field, enter: `Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Controllers.TumblrSharingController, Sitecore.Social.Tumblr.Client.Mvc.`
3. In the Parameters template field, enter: `/sitecore/templates/System/Social/Sharing/Tumblr Share Button Parameters`.

Track MVC Share button events

The MVC Share button events enable you to get the statistics of how many times people shared or liked a post on your website.

To track MVC Share button events:

1. Set up a Share button goal. However, you can use a ready goal, if you have it.
2. To subscribe to the click event of the specific Share button link, you must generate and pass the id to the view by extending the `ShareButtonViewModel` class. For example, use the following sample code:

```
namespace Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Model
{
  using Sitecore.Social.Client.Mvc.Areas.Social.Models;
  /// <summary>
  ///
  /// </summary>
  public class TumblrShareButtonViewModel : ShareButtonViewModel
  {
```

```
    /// <summary>
    /// Gets or sets the Tumblr share button identifier.
    /// </summary>
    /// <value>
    /// The Tumblr share button identifier.
    /// </value>
    public string TumblrShareButtonId { get; set; }
  }
}
```

3. Update the MVC sharing controller code. Compose a script that subscribes the code generated by the `SharingHelper.GetShareCallbackScript` method to the click event of the Tumblr Share button. For example, use the following sample code:

```
namespace Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Controllers
{
  using System;
  using System.Globalization;
  using System.Web.Mvc;
  using Sitecore.Common;
  using Sitecore.Globalization;
  using Sitecore.Social.Client.Common.Helpers;
  using Sitecore.Social.Client.Mvc.Areas.Social.Controllers;
  using Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Model;
  /// <summary>
  /// Tumblr sharing button controller.
  /// </summary>
  public class TumblrSharingController : SharingController
  {
    /// <summary>
    /// Gets Tweet button view.
    /// </summary>
    /// <returns>Tweet button view.</returns>
    public PartialViewResult TumblrButton()
    {
      var tumblrShareButtonId = string.Format("tumblrShare_{0}", Guid.NewGuid().ToID().ToShortID());
      var script = string.Format(
        CultureInfo.CurrentCulture,
        @"<script>document.getElementById('{0}').addEventListener('click', function(){{{1}}})</script>",
        tumblrShareButtonId,
        SharingHelper.GetShareCallbackScript(this.GoalName, Configuration.Settings.GetSetting("Social.Tumblr.DefaultShareGoal")))
      return this.PartialView(
        "~/Areas/Social/Views/Sharing/TumblrButton.cshtml",
        new TumblrShareButtonViewModel
        {
          SharePageUrlWithAnalyticsParameters = SharingHelper.GetSharePageUrlWithAnalyticsParameters(this.CampaignId),
          Text = Translate.Text(Common.Texts.ShareOnTumblr),
          SubscribeToShareButtonEventScript = script,
          SharePageUrl = SharingHelper.GetSharePageUrl(),
          TumblrShareButtonId = tumblrShareButtonId
        });
    }
  }
}
```

Note

For simplicity, this example uses Sitecore API to get a value of the `Social.Tumblr.DefaultShareGoal` setting. However, we recommend using Social Connected configuration framework.

1. Update the MVC sharing view code to enable the event tracking functionality. Update the @model to `Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Model.TumblrShareButtonViewModel`.
2. Set the share button link ID to `Model.TumblrShareButtonId`.
3. Render the `Model.SubscribeToShareButtonEventScript`.

Sample code

```
@model Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Model.TumblrShareButtonViewModel
<div style="float: right;">
  <a id="@Model.TumblrShareButtonId"
     href="http://www.tumblr.com/share/link?url=@Url.Encode(Model.SharePageUrlWithAnalyticsParameters)"
     title="@Model.Text"
     style="display: inline-block; text-indent: -9999px; overflow: hidden; width: 81px; height: 20px; background: url('https://platfor
    @Model.Text
  </a>
  <script type="text/javascript">
    (function (scriptTagId) {
      var tumblrScript = document.getElementById(scriptTagId);
      if (!tumblrScript) {
        tumblrScript = document.createElement("script");
        tumblrScript.src = "http://platform.tumblr.com/v1/share.js";
        tumblrScript.id = scriptTagId;
        document.body.appendChild(tumblrScript);
      }
    })('tumblrShareScript');
  </script>

  @Html.Raw(Model.SubscribeToShareButtonEventScript)
</div>
```

Send feedback about the documentation to docsite@sitecore.net.

Walkthrough: Configuring Share buttons using Sitecore sublayouts

You can configure and place Share buttons from different social networks on your website. When a website visitor clicks a Share button, an appropriate message with the link to the website is displayed in the social network. You can use either Sitecore sublayouts or Sitecore MVC renderings to create Share buttons, depending on the functionality you use on your website. In this walkthrough, Tumblr is used as an example.

Follow this walkthrough if you use Sitecore sublayouts.

This walkthrough describes how to:

- Create a Share button campaign
- Create a Share button goal
- Create a Share button parameters template
- Create a Share button sublayout
- Create a Share button sublayout markup
- Track sublayout Share button events

Create a Share button campaign

You can assign any campaign to a Share button. This feature uses Sitecore xDB. You can use standard Sitecore campaigns or create new ones for Social Connected.

To create a campaign for a Share button:

1. In the Content Editor, navigate to `/sitecore/system/Marketing Control Panel/Campaigns/Social` and click the relevant social network campaign group, for example, *Tumblr*.
2. Under the *Tumblr* category item, create a new item based on the *Campaign* template (`/sitecore/templates/System/Analytics/Campaign`). Name the new item, for example, *Tumblr Share Buttons*.

Note

Sitecore xDB requires a campaign to be deployed before you can use it. To deploy the new campaign, on the Home tab, click Edit and then click Deploy.

Create a Share button goal

Some social networks allow users to subscribe to Share button events. For example, Twitter's Tweet button has a 'tweet' event, which occurs when a user publishes a tweet using the Tweet button.

You can use standard Sitecore goals or create new ones for Social Connected. To create a goal for a Share button:

1. In the Content Editor, navigate to `/sitecore/system/Marketing Control Panel /Goals`.
2. In the *Goals* folder, create a new goal. Name it, for example, *Tumblr Share*.

Note

Sitecore xDB requires a goal to be deployed before you can use it. To deploy the new goal, on the **Home** tab, click **Edit** and then click **Deploy.**

Create a Share button parameters template

You must create a parameters template to pass the parameters to the share button controls, such as *Campaign* and *Goal*.

To create a parameters template for a Share button:

1. In the Content Editor, navigate to `/sitecore/templates/System/Social/Sharing`.
2. In the *Sharing* folder, create a new template with the following parameters:
   - Name – *Tumblr Share Button Parameters*
   - Base template – `/sitecore/templates/System/Social/Sharing/SharingControlParameters`
3. Create a *__Standard Values* item.
4. In the *__Standard Values* item, set the Goal and Campaign fields to the appropriate share button goal and campaign.

Create a Share button sublayout

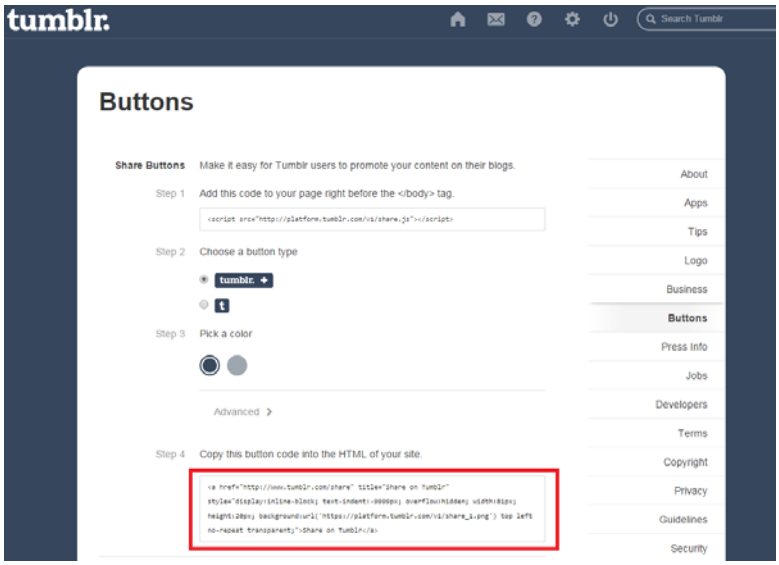
To create a share button sublayout:

1. In the Content Editor, navigate to `/sitecore/layout/Sublayouts/Social/Sharing`.
2. In the *Sharing* folder, create a new sublayout with the following parameters:
   - Name – the name of the new Share Button sublayout, for example,*Tumblr Button*
   - Item Location – `/sitecore/layout/Sublayouts/Social/Sharing`
   - File Location – `website/layouts/system/social/sharing`
3. Check the Create the associated C# code files check box.
4. In the *Tumblr Button* item, in the Parameters Template field, select the *Share Button Parameters* template that you just created.

Create a Share button sublayout markup

To create a sublayout markup of a share button, for example, for Tumblr:

1. Go to https://www.tumblr.com/buttons
2. Copy the code:



3. Open the `\website\layouts\system\social\sharing\TumblrButton.ascx` file.
4. In the `TumblrButton.ascx` file:
   - Paste the code that you copied from the Tumblr website.
   - Replace the value of the `href` attribute with:

```
http://www.tumblr.com/share/link?url=<%= HttpUtility.UrlEncode(SharingHelper.GetSharePageUrlWithAnalyticsParameters(this.Campaign
```

   - Replace the value of the `title` attribute and the element contents with:

```
<%= Translate.Text(Texts.ShareOnTumblr) %>
```

   - Add the following Tumblr script:

```
<script type="text/javascript">
  (function (scriptTagId) {
    var tumblrScript = document.getElementById(scriptTagId);
    if (!tumblrScript) {
      tumblrScript = document.createElement("script");
      tumblrScript.src = "http://platform.tumblr.com/v1/share.js";
      tumblrScript.id = scriptTagId;
      document.body.appendChild(tumblrScript);
    }
  })('tumblrShareScript');
</script>
```

Sample code

In the sample code below, the `href` and `title` attributes have been set in the code behind.

The `TumblrButton.ascx` file:

```
<%@ Control Language="C#" AutoEventWireup="true" Inherits="Sitecore.Social.Tumblr.Client.Sharing.Controls.TumblrButton" %>
<div style="float: right;" runat="server">
  <a
    runat="server"
    ID="tumblrShareButton"
    style="display: inline-block; text-indent: -9999px; overflow: hidden; width: 81px; height: 20px; background: url('https://platform
  </a>
</div>
<script type="text/javascript">
  (function (scriptTagId) {
    var tumblrScript = document.getElementById(scriptTagId);
    if (!tumblrScript) {
      tumblrScript = document.createElement("script");
      tumblrScript.src = "http://platform.tumblr.com/v1/share.js";
      tumblrScript.id = scriptTagId;
      document.body.appendChild(tumblrScript);
    }
  })('tumblrShareScript');
</script>
```

The code-behind `TumblrButton.ascx.cs` file:

```
namespace Sitecore.Social.Tumblr.Client.Sharing.Controls
{
  using System;
  using System.Globalization;
  using System.Web;
  using System.Web.UI.HtmlControls;
  using Sitecore.Globalization;
  using Sitecore.Social.Client.Common.Helpers;
  using Sitecore.Social.Client.Sharing.Controls;
  /// <summary>
  /// Tweet button control.
  /// </summary>
  public class TumblrButton : ShareButtonBase
  {
    /// <summary>
    /// The Tumblr share button
    /// </summary>
    protected HtmlAnchor tumblrShareButton;
    /// <summary>
    /// Handles the Load event of the Page control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="System.EventArgs"/> instance containing the event data.</param>
    protected void Page_Load(object sender, EventArgs e)
    {
      if (!this.IsPostBack)
      {
        this.tumblrShareButton.Title = Translate.Text(Common.Texts.ShareOnTumblr);
        this.tumblrShareButton.HRef = string.Format("http://www.tumblr.com/share/link?url={0}", HttpUtility.UrlEncode(SharingHelper.Ge
        this.tumblrShareButton.InnerText = Translate.Text(Common.Texts.ShareOnTumblr);
      }
    }
  }
```

```
}
```

Note

The button class inherits from the `Sitecore.Social.Client.Sharing.Controls.ShareButtonBase` class. This base class manipulates rendering parameters.

Track sublayout Share button events

You can track Share button events to get the statistics of how many times people shared or liked a post on your website.

Update the `TumblrButton.ascx.cs` file with this code to enable event tracking:

```
namespace Sitecore.Social.Tumblr.Client.Sharing.Controls
{
  using System;
  using System.Globalization;
  using System.Web;
  using System.Web.UI.HtmlControls;
  using Sitecore.Globalization;
  using Sitecore.Social.Client.Common.Helpers;
  using Sitecore.Social.Client.Sharing.Controls;
  /// <summary>
  /// Tweet button control.
  /// </summary>
  public class TumblrButton : ShareButtonBase
  {
    /// <summary>
    /// The Tumblr share button
    /// </summary>
    protected HtmlAnchor tumblrShareButton;
    /// <summary>
    /// Handles the Load event of the Page control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="System.EventArgs"/> instance containing the event data.</param>
    protected void Page_Load(object sender, EventArgs e)
    {
      if (!this.IsPostBack)
      {
        this.tumblrShareButton.Title = Translate.Text(Common.Texts.ShareOnTumblr);
        this.tumblrShareButton.HRef = string.Format("http://www.tumblr.com/share/link?url={0}", HttpUtility.UrlEncode(SharingHelper.Ge
        this.tumblrShareButton.InnerText = Translate.Text(Common.Texts.ShareOnTumblr);
      }
      var script = string.Format(
        CultureInfo.CurrentCulture,
        @"<script>document.getElementById('{0}').addEventListener('click', function(){{{1}}})</script>",
        this.tumblrShareButton.ClientID,
        SharingHelper.GetShareCallbackScript(this.GoalName, Configuration.Settings.GetSetting("Social.Tumblr.DefaultShareGoal")));
      this.Page.ClientScript.RegisterStartupScript(this.GetType(), "tumblrsubscribe" + this.tumblrShareButton.ClientID, script);
    }
  }
}
```

Note

When a visitor clicks the share button, a goal that is set in the rendering parameters will be triggered. If no goal has been set, the default goal will be used. This is specified in the `Social.Tumblr.DefaultShareGoal Sitecore` setting:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <!-- SOCIAL - DEFAULT TUMBLR GOAL
           Default goal to trigger on Tumblr button.
           Default value: Tumblr Share
```

```
    -->
  <setting name="Social.Tumblr.DefaultShareGoal" value="Tumblr Share"/>
```

For simplicity, this example uses Sitecore API to get a value for the `Social.Tumblr.DefaultShareGoal Sitecore` setting. However, you should use the Social Connected configuration framework.

Send feedback about the documentation to docsite@sitecore.net.

Walkthrough: Configuring Social Connected for posting messages

To enable marketers to post content messages in the Social Connected module, you must configure the module to access social networks. When interacting with social networks, the module uses the API of the social networks.

You must create a social network application for every social network that you want the module to communicate with, create a Sitecore item for the social network application, and create a Sitecore item for the social network account.

This walkthrough outlines how to:

- Create a social network application
- Create a Sitecore item for the social network application
- Create an account folder
- Create a social network account item
- Assign security permissions to an author
- Delete a social network account item

Create a social network application

Before you can link your website to a social network, you must first create an application. The Social Connected module receives visitors' profile information from a social network via the social network application.

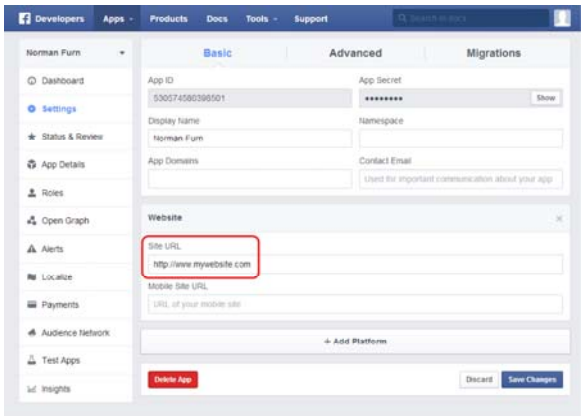
You can create applications and link to the following social networks:

- Facebook
- Twitter
- LinkedIn
- Google+

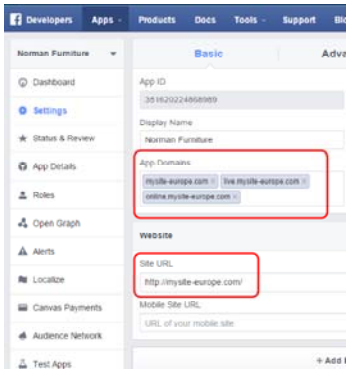
Create a Facebook application

To create a Facebook application:

1. Go to https://developers.facebook.com/apps and create an application on the website platform.
2. In the Facebook application, fill in all the required fields.
3. Add a website platform.
4. Enter the URL of the webpage where you want to place the login controls.



5. In the App Domains field, enter all the CD and CM domains, and in the Site URL field, enter the CM domain:



If your Sitecore instance uses a scaled environment, Social Connector requires that the base domain for all the Content Delivery (CD) and Content Management (CM) servers is the same.

6. On the pane on the left, click Status & Review, and click the switch to make your app publicly available.

Facebook Business Manager

Note

You can use this functionality of Social Connected starting from Sitecore XP 8 Update-2.

In case you plan to assign different Facebook applications to different Facebook login controls, you should perform additional configuration.

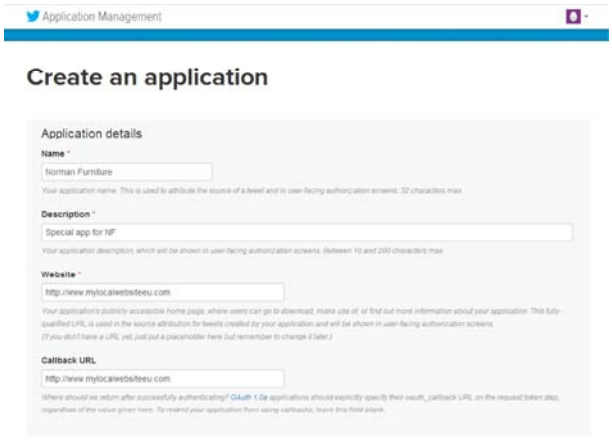
Starting from version 2.0, Facebook Graph API introduces application-scoped user IDs, which means that an ID of the same person is different between Facebook applications.

To allow Social Connected properly match users across different applications, you should set up the Business Manager and link it to all Facebook applications you are planning to use on your site. For more information on how to set up the Business Manager, visit the following website: https://business.facebook.com/

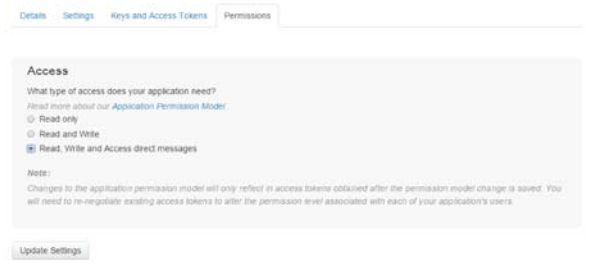
Create a Twitter application

To create a Twitter application:

1. Go to https://dev.twitter.com/apps and create a new application.
2. In the Twitter application, fill in all the required fields.



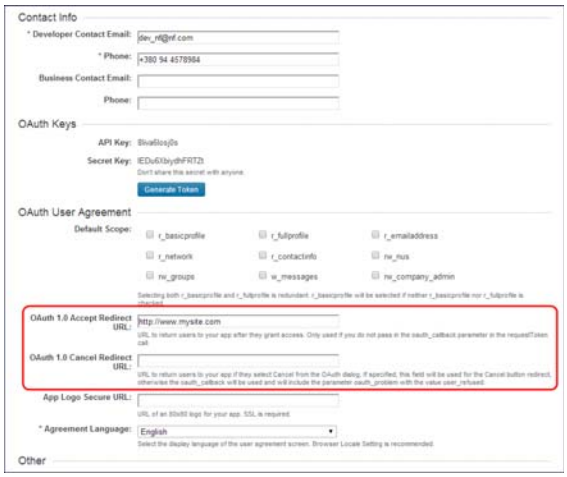
3. Enter the URL of the webpage where you want to place the login controls.
4. Set the application permissions to *Read, Write and Access direct messages*.



Create a LinkedIn application

To create a LinkedIn application:

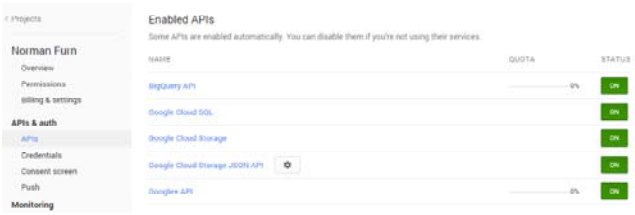
1. Go to https://www.linkedin.com/secure/developer and create a new application.
2. In the LinkedIn application, fill in all the required fields.
3. Enter the URL of the webpage where you want to place the login controls.



Create a Google+ application

To create a Google+ application:

1. Go to https://console.developers.google.com/project and create a new project.
2. In the Google+ project, enable Google+ API.



3. On the Credentials page, create a new client ID.
4. In the Create Client ID dialog box:
   - In the Authorized Javascript origins field, enter *http://hostname*
   - In the Authorized redirect URI field, enter:

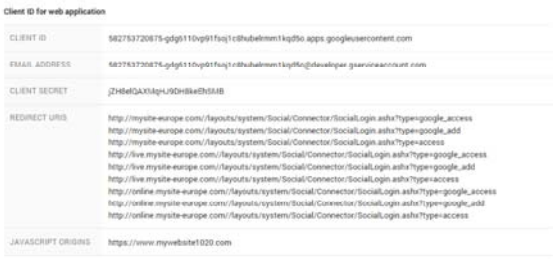
     http://hostname/layouts/system/Social/Connector/SocialLogin.ashx?type=google_access

     http://hostname/layouts/system/Social/Connector/SocialLogin.ashx?type=google_add

     http://hostname/layouts/system/Social/Connector/SocialLogin.ashx?type=access

     where *hostname* is the name of the host that is running the website.

   If your Sitecore instance uses a scaled environment, in the Authorized redirect URI field, enter the same URIs replacing *hostname* with all the CD and CM domains, for example:



5. On the Credentials page, in the Public API access section, click Create new Key.

Create a Sitecore item for the social network application

After you have created a social network application, you must create a Sitecore item for it. This Sitecore item contains connection keys that Sitecore uses to work with the social network application.

To create a Sitecore item for the social network application:

1. In Sitecore, in the Content Editor, open the *sitecore/system/social/applications/default* folder.
2. Create an item based on the *sitecore/templates/system/social/application* template.

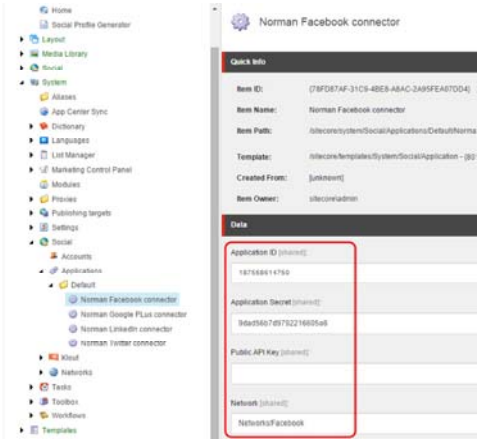
   Important

   The *sitecore/system/social/applications/default* folder must contain only one item for a social network. If you want to use several web applications for one social network, you must create new items for each web application in the *sitecore/system/social/applications* folder.

3. In the new item, fill in the Application Id, Application Secret, Public API key (for Google+ only) and Network fields:

Field	Value

Application Id	Facebook – the App ID value Twitter – the API Key value LinkedIn – the API Key value Google+ – the Client ID value
Application Secret	Facebook – the App Secret value Twitter – the API Secret value LinkedIn – the Secret Key value Google+ – the Client Secret value
Public API key	Google+ – the API Key value
Network	Networks/Facebook Networks/Twitter Networks/LinkedIn Networks/Google+



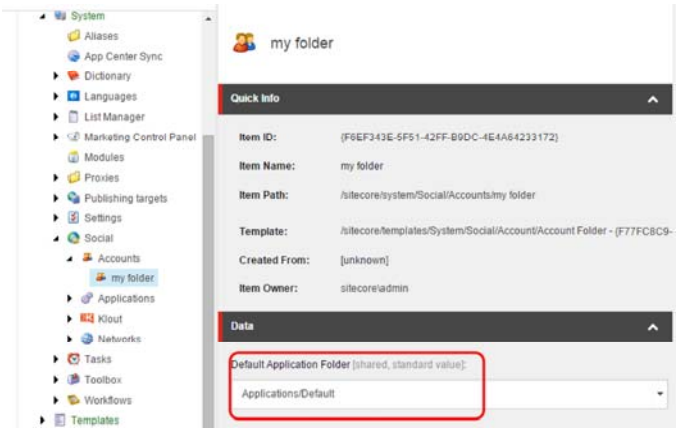
Create an account folder

Before you create an account item, you can create an account folder to group your account items.

To create an account folder:

1. In Sitecore, in the Content Editor, open the *sitecore/system/social/accounts* folder.
2. Click the Account Folder insert option, and enter the name of the folder.
3. In the new folder, on the Content tab, in the Default Application Folder field, specify the application folder.

   When you create an account item from this folder, the Add a Social Network Account wizard only displays applications from this application folder.



Create a social network account item

A social network account item stores the credentials of the social network account that you post messages to. For example, this could be a company account or a personal account. If the messages for this account need to be reviewed before posting, a social network account item stores review settings.

To create a social network account item:

1. In the Content Editor, open the *sitecore/system/social/accounts* folder, and select an account folder.
2. Click the Facebook Account, Twitter Account, LinkedIn Account, Google+ Account or Social Marketer Account insert option.

The Create a Social Network Account wizard guides you through the process of creating a social network account item.

The following table shows the most relevant settings to create an account item using the wizard:

Wizard page	Description
Social Network Application	Select the social network application that the account uses to cooperate with the social network.
Authentication	Enter the credentials of the social network user account. The module posts messages to the timeline or pages of the specified user. By default, an authentication session timeout is set to 60 seconds. During this period, the user can log in to the social network. If the user does not log in to the social network within this timeframe, the wizard stops. You can change this default timeout parameter in the `social.config` file, in the `Social.AddNetworkAccountWizard.AuthenticationTimeout` setting. To create an account, you must accept all the permission requests from the social network application.
Review	If you do not want the messages to go through the review process before posting, skip this page. If you want the messages to be reviewed, select a workflow to place the messages to, and assign roles for the reviewer.

If you use the default *Social Message* workflow, we recommend that you assign the following security roles to the reviewer:

Messages	Reviewer Role
Facebook	*sitecore\Social Message Workflow Reviewer* *sitecore\Author* *sitecore\Facebook Message Reviewer*
Twitter	*sitecore\Social Message Workflow Reviewer* *sitecore\Author* *sitecore\Twitter Message Reviewer*
Facebook and Tweeter	*sitecore\Social Message Workflow Reviewer* *sitecore\Author* *sitecore\Facebook Message Reviewer* *sitecore\Twitter Message Reviewer*

If you use the default *Social Marketer Message* workflow, we recommend that you assign the *sitecore\Social Marketer Message Reviewer* role.

If you use your customized workflow, use the security roles that you created for this workflow.

Assign security permissions to an author

To let the authors create content messages, you must make them members of the following security roles:

- *sitecore\Author*

  Permissions to the Content Editor, Workbox, and the Messages dialog box.

- *sitecore\Social Message Author*

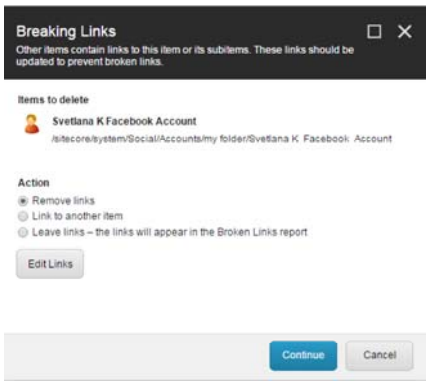
  Permissions to create, edit, and remove all messages

- *sitecore\Social Message Workflow Editor*

  This role is required if an author works with the messages in the *Social Message* workflow. Permissions to use the *Draft* state and its commands in the *Social Message* workflow.

Delete a social network account item

You can delete a network account item or a social marketer account item in the same way that you delete a Sitecore item. When you click Delete the item, the Breaking Links dialog box appears:



If the item that you want to delete has attached messages, these messages are not deleted. However, the Link to another item option does not link the existing messages to another item. If you want to use existing messages, you must create new messages and move the content from the existing messages to the new ones.

Send feedback about the documentation to docsite@sitecore.net.

Walkthrough: Configuring Social Connected in a scaled environment

If your Sitecore solution works in a scaled environment, you must configure the Social Connected module to work in this environment.

Prerequisites:

- The Sitecore solution must be configured according to the Sitecore scaling recommendations.
- The CM and CD servers must have access to the social network services over the Internet.
- The Core database should be shared between all the CM and CD servers or data replication should be used instead.

This walkthrough outlines how to:

- Disable Social Connected configuration files on the Processing and Reporting servers.
- Disable indexes specific to the CM servers.
- Disable the Experience Profile plugin on the CD servers.
- Enable scalability.
- Set the URL that is used in messages.
- Set up the scheduling activities.

Disable Social Connected configuration files on the Processing and Reporting Servers

On all Processing and Reporting servers:

1. In the `\Website\App_Config\Include\Social` folder, find all the configuration files.
2. Rename these files to `[file_name].config.disabled`.

Disable indexes specific to the CM servers

On all the CD servers:

1. In the `\Website\App_Config\Include\Social` folder, find all the configuration files with the `Index.Master` suffix.
2. Rename these files to `[file_name].config.disabled`

Disable the Experience Profile plugin on the CD servers

On all the CD servers:

1. In the `\Website\App_Config\Include\Social` folder, find the `Sitecore.Social.ExperienceProfile.config` file.
2. Rename it to `Sitecore.Social.ExperienceProfile.config.disabled`.

Enable scalability

On all the CD and CM servers:

1. In the `\Website\App_Config\Include\Social` folder, rename the `Sitecore.Social.ScalabilitySettings.config.disabled` file to `Sitecore.Social.ScalabilitySettings.config`.
2. In the file, set the value of the `Social.Scalability.ContentManagement.TasksExecutionInstance` setting to the name of the CM server that performs all the scheduled activities. This value must be the same on all the servers because only one CM server must perform the scheduling activities.

Set the URL that is used in messages

On all the CM servers:

1. In the `\Website\App_Config\Include\Social` folder, find the `Sitecore.Social.config` file.
2. Set the value of the `Social.LinkDomain` setting to the host name.

Set up the scheduling activities (optional)

On all the CM and CD servers except for the CM server that performs the scheduled activities:

1. In the `\Website\App_Config\Include\Social` folder, find the `Sitecore.Social.config` file.
2. In the `scheduling` section, for all four agents, set the interval parameter to `00:00:00`.
3. If you have enabled `Sitecore.Social.Klout.config`, in the scheduling section, in the `Sitecore.Social.Klout.Client.Tasks.SynchronizeKloutScores` agent, set the interval parameter to `00:00:00`.

Send feedback about the documentation to docsite@sitecore.net.

Walkthrough: Configuring Social Connector to work with a social network

After you have installed the Social Connected module, you must configure the Social Connector before website visitors can log in to your website using social network credentials.

This walkthrough outlines how to:

- Create a social network application
- Create a Sitecore item for the social network application
- Configure access to the visitor's profile information
- Request approval for Facebook application permissions
- Configure and add the social network log-in control to your website
- Improve the performance of Social Connector

Create a social network application

Before you can link your website to a social network, you must first create an application. The Social Connected module receives visitors' profile information from a social network via the social network application.

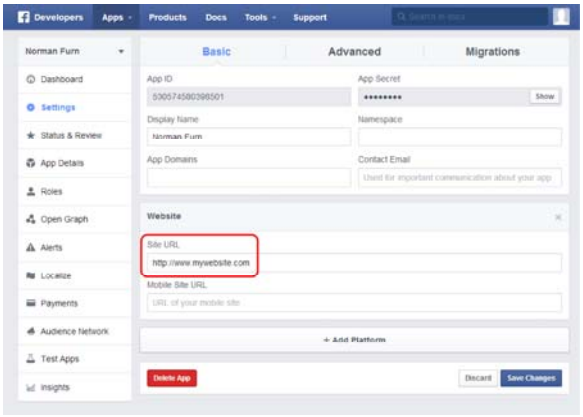
You can create applications and link to the following social networks:

- Facebook
- Twitter
- LinkedIn
- Google+

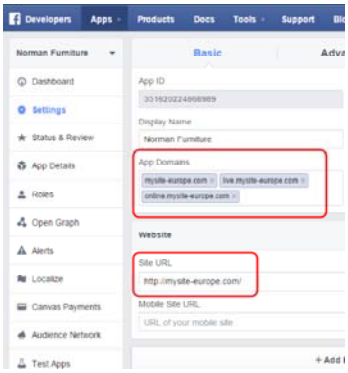
Create a Facebook application

To create a Facebook application:

1. Go to https://developers.facebook.com/apps and create an application on the website platform.
2. In the Facebook application, fill in all the required fields.
3. Add a website platform.
4. Enter the URL of the webpage where you want to place the login controls.



5. In the App Domains field, enter all the CD and CM domains, and in the Site URL field, enter the CM domain:



If your Sitecore instance uses a scaled environment, Social Connector requires that the base domain for all the Content Delivery (CD) and Content Management (CM) servers is the same.

6. On the pane on the left, click Status & Review, and click the switch to make your app publicly available.

Facebook Business Manager

Note

You can use this functionality of Social Connected starting from Sitecore XP 8 Update-2.

In case you plan to assign different Facebook applications to different Facebook login controls, you should perform additional configuration.

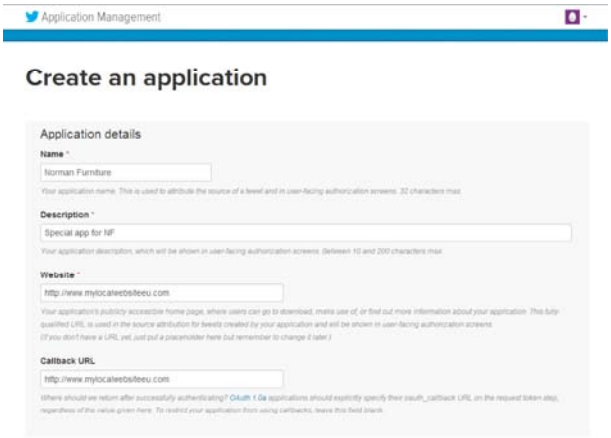
Starting from version 2.0, Facebook Graph API introduces application-scoped user IDs, which means that an ID of the same person is different between Facebook applications.

To allow Social Connected properly match users across different applications, you should set up the Business Manager and link it to all Facebook applications you are planning to use on your site. For more information on how to set up the Business Manager, visit the following website: https://business.facebook.com/

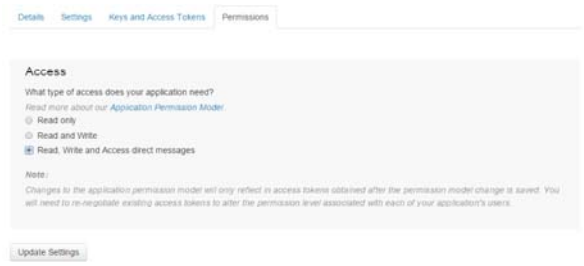
Create a Twitter application

To create a Twitter application:

1. Go to https://dev.twitter.com/apps and create a new application.
2. In the Twitter application, fill in all the required fields.



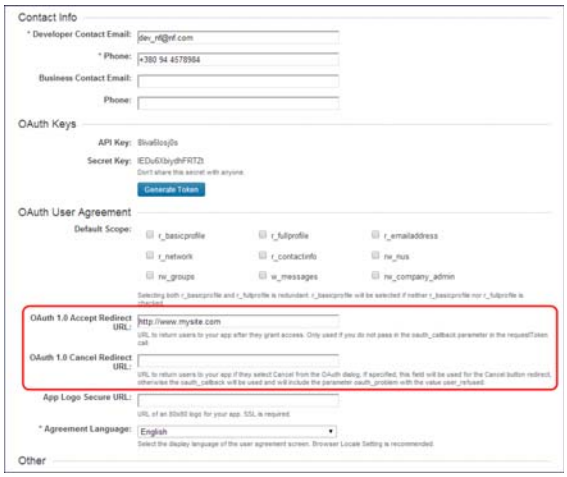
3. Enter the URL of the webpage where you want to place the login controls.
4. Set the application permissions to *Read, Write and Access direct messages*.



Create a LinkedIn application

To create a LinkedIn application:

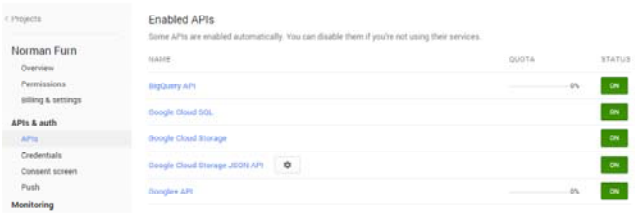
1. Go to https://www.linkedin.com/secure/developer and create a new application.
2. In the LinkedIn application, fill in all the required fields.
3. Enter the URL of the webpage where you want to place the login controls.



Create a Google+ application

To create a Google+ application:

1. Go to https://console.developers.google.com/project and create a new project.
2. In the Google+ project, enable Google+ API.



3. On the Credentials page, create a new client ID.
4. In the Create Client ID dialog box:
   - In the Authorized Javascript origins field, enter *http://hostname*
   - In the Authorized redirect URI field, enter:

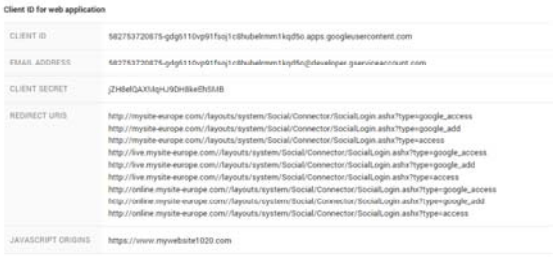
     http://hostname/layouts/system/Social/Connector/SocialLogin.ashx?type=google_access

     http://hostname/layouts/system/Social/Connector/SocialLogin.ashx?type=google_add

     http://hostname/layouts/system/Social/Connector/SocialLogin.ashx?type=access

     where *hostname* is the name of the host that is running the website.

   If your Sitecore instance uses a scaled environment, in the Authorized redirect URI field, enter the same URIs replacing *hostname* with all the CD and CM domains, for example:



5. On the Credentials page, in the Public API access section, click Create new Key.

Create a Sitecore item for a social network application

After you have created a social network application, you must create a Sitecore item for it. This Sitecore item contains connection keys that Sitecore uses to work with the social network application.

To create a Sitecore item for the social network application:

1. In Sitecore, in the Content Editor, open the *sitecore/system/social/applications/default* folder.
2. Create an item based on the *sitecore/templates/system/social/application* template.

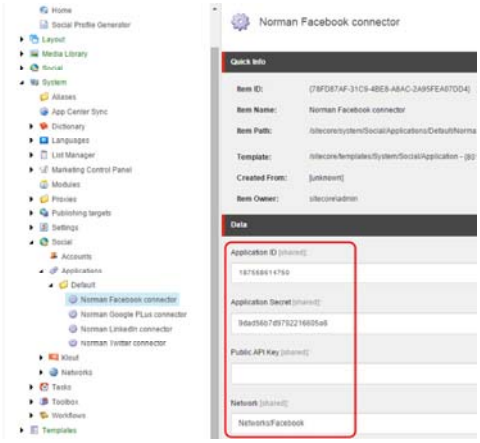
   Important

   The *sitecore/system/social/applications/default* folder must contain only one item for a social network. If you want to use several web applications for one social network, you must create new items for each web application in the *sitecore/system/social/applications* folder.

3. In the new item, fill in the Application Id, Application Secret, Public API key (for Google+ only) and Network fields:

Field	Value

Application Id	Facebook – the App ID value Twitter – the API Key value LinkedIn – the API Key value Google+ – the Client ID value
Application Secret	Facebook – the App Secret value Twitter – the API Secret value LinkedIn – the Secret Key value Google+ – the Client Secret value
Public API key	Google+ – the API Key value
Network	Networks/Facebook Networks/Twitter Networks/LinkedIn Networks/Google+



Configure access to a visitor's profile information

The first time a visitor logs in to your website with their social network credentials, the social network asks the visitor to grant your website access to the information on their social network profile. You can configure a set of fields that the social network application requests access to.

The following image, shows an example of the dialog box that a social network user sees the first time they log in to the website with their social network credentials:

To set the list of fields that the social network application can access in the visitor's profile on a social network:

1. Open the `[website_root]\app_config\include\social\sitecore.social.profilemapping.[network_name].config` file.
2. In the configuration section for the social network, in the configuration section for the field that you want the social network application to use, set the `field enabled` parameter to true.
3. If the field is private, you must set its permission parameter according to the official social network documentation.

Important

Do not disable the fields that are enabled by default because this might affect the way that they are presented on your website.

For example, to get access to the Groups' field on Facebook, the field section in the `Sitecore.social.profilemapping.Facebook.config` file should look like this:

```
<!-- 'Groups' field. -->
<!-- The Groups that the user belongs to. -->
<field enabled="true" originalKey="" sitecoreKey="fb_groups" permission="user_groups" access="/me/groups" text="Groups" />
```

For more information about user profile fields, see the official documentation of the social network:

- Facebook documentation
- Twitter documentation
- LinkedIn documentation
- New Google+ API

Request approval for Facebook application permissions

This information is only relevant to Facebook applications.

After you have set the list of fields that the Facebook application can access in the Facebook visitor's profile, you must get Facebook approval for permissions. The permissions relate to a field or group of fields. Facebook requires approval for some permissions before you can get access to the fields with those permissions.

For the Social Connected module and the Experience Profile application to work correctly, a Facebook application must have access to the following permissions:

Permission	Description	Requires Facebook approval
public_profile	Used in the Social Connector, personalization, and in the Experience Profile Gives access to a subset of a user's public profile fields: • `id` • `name` • `first_name` • `last_name` • `link` • `gender` • `locale` • `age_range`	No
user_friends	Used in the Social Connector	No
email	Used in the Social Connector and personalization	No
user_birthday	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_education_history	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_interests	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_likes	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_location	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_relationships	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_religion_politics	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_website	Used in the Social Connector, personalization, and in the Experience Profile	Yes
user_work_history	Used in the Social Connector, personalization, and in the Experience Profile	Yes
read_stream	Used in the Social Connector and personalization	Yes
publish_actions	Used in content posting and goal posting	Yes
manage_pages	Used in content posting	Yes

The other fields that you enable in the `Sitecore.Social.ProfileMapping.Facebook.config` file might require Facebook approval.

You can read more about permissions with Facebook login on Facebook's website.

You can also find more information about Facebook's review guidelines on their website.

Configure and add the social network log-in control

After you have created a social network application and a Sitecore item for it, and you have configured access to the visitor's profile information, you can configure and add a social network log-in control to your website. The visitors use this control to log in to your website.

You can add log-in controls in the same way that you add new components on the webpage. The controls are located in the *sitecore/layout/sublayouts/social/connector* folder. For the MVC layout, use the Social Connected renderings in the *sitecore/layouts/Renderings/Social MVC/Connector* folder. You can customize this control by changing images, texts, and so on.

By default, the social network control uses the social web application that is stored in the *sitecore/system/social/applications/default* folder.

To set a non-default social network application for the social network control in the UI:

1. In the Content Editor, select the item that you want to place a login control on.
2. On the ribbon, in the Presentation tab, click Details.
3. In the Layout details dialog box, add the *Login with <social network>* control.
4. Click the control and in the Application field, select the application that you want the control to use.

To set a non-default social network application for the social network control using the API, pass the Application Key and Application Secret values of the application to the API using the `ApplicationCredentials` class.

To set the webpage that the visitor is redirected to after they have entered social network credentials and granted access to their profile information:

1. In the Content Editor, select the item presentation details.
2. Add a Social Connector control.
3. Edit the control.
4. In the Callback URL field, enter the URL of the webpage.

   If this field is blank, the module uses the URL of the current item.

Improve the performance of Social Connector

After you have performed all the required configuration actions, you can reduce the time that the Social Connector spends on finding an existing user when the website visitor logs in to your website. To do this, in the `web.config` file, in the `configuration/system.web/profile/properties` section, paste the following string:

```
<add type="System.String" name="SC_SocialNetworkId" />
```

Send feedback about the documentation to docsite@sitecore.net.

Walkthrough: Configuring Social Messaging

When you publish Sitecore items, Social Connected can post messages to social networks. You bind the message to the content item, and the module posts this message automatically when you publish the content item. You can also post these messages manually without publishing the content items.

This walkthrough describes how to configure the Social Messaging feature. The Tumblr social network is used as an example.

The walkthrough outlines how to:

- Create a new Account wizard
- Create a social message template
- Create the campaign items for social network messages
- Create a social network message reviewer role
- Implement the new social network classes
- Update the social network configuration file

Create a new Account wizard

To be able to create items for social network accounts, you must create a new Account wizard.

Note

The Tumblr social network allows one user account to have multiple blogs. In the following example, posting is implemented only to the primary blog. To be able to select the blog where you want to post messages when creating a network account, you need to implement a custom wizard. Use the Facebook Account wizard as an example.

To create a wizard for a new social network account:

1. In the Content Editor, navigate to `sitecore/templates/branches/system/social` and in the *Social* folder create a new item, based on the *Command* template (`sitecore/templates/system/branches/command`). You can use the *Command Template* insert option in the *Social* folder, or just duplicate the existing item for another network. Name it, for example, *Tumblr Account.*
2. In the new item, in the Command field, enter *social:account:add(network=Tumblr)*.
3. Navigate to `/sitecore/templates/System/Social/Account/Account Folder`.
4. In the *__Standard Values* subitem, in the Insert options field, add the `/sitecore/templates/Branches/System/Social/Tumblr Account` item.

As a result, the new Account wizard (for example, the Tumblr Account wizard), is added to the *Accounts* folder in `/sitecore/system/Social/Accounts`.

Create a social message template

Social messages are stored in the message items that are based on the social message template.

To create a social message template:

1. In the Content Editor, navigate to `/sitecore/templates/System/Social/Message` and in the *Message* folder, create a new template based on the *Message* template.
2. In the Data section, add the data fields that the social network message should contain, for example, a Title field for a Tumblr blog post. The fields should be marked as Shared.

Create the campaign items for social network messages

By default, a message contains a link. When a user clicks the link, it triggers the relevant campaign. Campaigns are stored in the campaign items.

To create the campaign items for social network messages:

1. In the Content Editor, navigate to `/sitecore/system/Marketing Control Panel/Campaigns/Social` and in the *Social* folder, create a new item based on the *Campaign Category* template (`/sitecore/templates/System/Analytics/Campaign Category`). Name the new item, for example, *Tumblr*.
2. In the new item, create the following new subitems:
   - A new subitem based on the `/sitecore/templates/System/Analytics/Campaign Category` template. Name the new item, for example, *Tumblr Message Campaigns*. This item will store the custom campaigns that will use the social messages.
   - A new subitem based on the`/sitecore/templates/System/Analytics/Campaign` template. Name the new item, for example, *Tumblr Content Messages*. This is the default campaign for content messages.
   - A new subitem based on the `/sitecore/templates/System/Analytics/Campaign` template. Name the new item, for example, *Tumblr Goal Messages*. This is the default campaign for goal messages.

Note

The xDB requires a campaign to be deployed before you can use it. To deploy the new campaign, on the Home tab, click Edit and then click Deploy.

Create a social network message reviewer role

If you use social message workflows, then you should create a reviewer role specific to the social network. The social message workflows are parts of the default workflows of Social Connected.

To create a message reviewer role for a social network, for example, for Tumblr:

1. In the Role Manager, click New.
2. In the New Role dialog box, enter a name for the role, for example, *Tumblr Message Reviewer*.
3. In the Domain field, select *sitecore*.

When you create a network account, you will be able to select this new role from the list.

Implement the new social network classes

The following sections describe how to implement the new classes required to configure social messaging, and include code samples for each class.

Message Item

Implement a class that inherits from the `Sitecore.Social.MessagePosting.Items.MessageItemBase` class. It represents the Sitecore message item for the specific social network.

The following code illustrates how to implement the `TumblrMessageItem` class for Tumblr (*Sitecore.Social.Tumblr* project):

```
namespace Sitecore.Social.Tumblr.MessagePosting.Items
{
  using Sitecore.Data.Items;
  using Sitecore.Social.MessagePosting.Items;
  /// <summary>
  /// Tumblr message item.
  /// </summary>
  public class TumblrMessageItem : MessageItemBase
  {
    /// <summary>
    /// Initializes a new instance of the <see cref="TumblrMessageItem"/> class.
    /// </summary>
    public TumblrMessageItem()
    {
    }
    /// <summary>
    /// Initializes a new instance of the <see cref="TumblrMessageItem"/> class.
    /// </summary>
    /// <param name="item">The item.</param>
    public TumblrMessageItem(Item item)
      : base(item)
    {
    }
    /// <summary>
    /// Gets the value of the Title field.
    /// </summary>
    /// <value>
    /// The title.
    /// </value>
    public string Title
```

```
    {
      get
      {
        var titleField = this.MessageItem.Fields[Common.ItemIDs.TitleFieldId];
        return titleField != null ? titleField.Value : null;
      }
    }
  }
}
```

Message Data

Implement a class that represents the social network message data entity. Use the `Sitecore.Social.Domain.Model.MessageData` class as a base class.

The following code illustrates how to implement the `TumblrMessageData` class for Tumblr (*Sitecore.Social.Tumblr* project):

```
namespace Sitecore.Social.Tumblr.Model
{
  using System;
  using Sitecore.Social.Domain.Model;
  /// <summary>
  /// The Tumblr messsage data.
  /// </summary>
  [Serializable]
  public class TumblrMessageData : MessageData
  {
    /// <summary>
    /// Gets or sets the title.
    /// </summary>
    /// <value>
    /// The title.
    /// </value>
    public string Title { get; set; }
  }
}
```

Network Provider Message

Implement a class that represents a social message for the network provider. Use the `Sitecore.Social.NetworkProviders.Messages.Message` class as a base class.

The following code illustrates how to implement the `TumblrMessage` class for Tumblr (*Sitecore.Social.Tumblr* project):

```
namespace Sitecore.Social.Tumblr.Networks.Messages
{
  using Sitecore.Social.NetworkProviders.Messages;
  /// <summary>
  /// The Tumblr message.
  /// </summary>
  public class TumblrMessage : Message
  {
    /// <summary>
    /// Gets or sets the title.
    /// </summary>
    /// <value>
    /// The title.
    /// </value>
    public string Title { get; set; }
  }
}
```

Message Data Builder

Implement a class that is responsible for operating the message-related classes. Use the `Sitecore.Social.Builders.MessageDataBuilder` class as a base class. You must implement two `BuildFrom` method overloads:

- `MessageData BuildFrom(MessageItemBase messageItemBase)`– responsible for building the `MessageData` instance from the `MessageItemBase` instance.
- `NetworkProviders.Messages.Message BuildFrom(MessageData messageData)`– responsible for building the network provider, a specific message class instance from the `MessageData` instance.

Note

If you have added additional fields to the message template, you must override the `UpdateMessageItem` method. The method is responsible for updating Sitecore items of social messages, taking an appropriate inheritor of the `Sitecore.Social.Domain.Model.MessageData` class as a parameter.

The following code illustrates how to implement the `TumblrMessageDataBuilder` class for Tumblr (`Sitecore.Social.Tumblr` project):

```
namespace Sitecore.Social.Tumblr.Builders
{
  using Sitecore.Data;
  using Sitecore.Data.Items;
  using Sitecore.Social.Builders;
  using Sitecore.Social.Domain.Model;
  using Sitecore.Social.Infrastructure.Extensions;
  using Sitecore.Social.Infrastructure.Identifiers;
  using Sitecore.Social.MessagePosting.Items;
  using Sitecore.Social.Tumblr.MessagePosting.Items;
  using Sitecore.Social.Tumblr.Model;
  using Sitecore.Social.Tumblr.Networks.Messages;
  /// <summary>
  /// Tumblr message data builder.
  /// </summary>
  public class TumblrMessageDataBuilder : MessageDataBuilder
  {
    /// <summary>
    /// Builds from social message base.
    /// </summary>
    /// <param name="messageItemBase">The social message base.</param>
    /// <returns>
    /// The <seealso cref="T:Sitecore.Social.Domain.Model.MessageData" />
    /// </returns>
    public override MessageData BuildFrom(MessageItemBase messageItemBase)
    {
      var tumblrMessageItem = (TumblrMessageItem)messageItemBase;
      return new TumblrMessageData()
      {
        CampaignId = string.IsNullOrEmpty(tumblrMessageItem.CampaignId) ? IDIdentifier.Empty : (new ID(tumblrMessageItem.CampaignId)).(
        CreatedDate = tumblrMessageItem.CreatedDate,
        PostedDate = tumblrMessageItem.PostedDate,
        Title = tumblrMessageItem.Title,
        Text = tumblrMessageItem.Message,
        Link = tumblrMessageItem.Link
      };
    }
    /// <summary>
    /// Builds from social message base.
    /// </summary>
    /// <param name="messageData">The social message base.</param>
    /// <returns>
    /// The <seealso cref="T:Sitecore.Social.NetworkProviders.Messages.Message" />
    /// </returns>
    public override NetworkProviders.Messages.Message BuildFrom(MessageData messageData)
    {
      var tumblrMessageData = (TumblrMessageData)messageData;
      return new TumblrMessage()
      {
        Text = tumblrMessageData.Text,
        Title = tumblrMessageData.Title
```

```
      };
    }
    /// <summary>
    /// Updates the message item.
    /// </summary>
    /// <param name="messageItem">The message item.</param>
    /// <param name="messageData">The message data.</param>
    public override void UpdateMessageItem(Item messageItem, MessageData messageData)
    {
      base.UpdateMessageItem(messageItem, messageData);
      var titleField = messageItem.Fields[Common.ItemIDs.TitleFieldId];
      if (titleField != null)
      {
        titleField.Value = ((TumblrMessageData)messageData).Title;
      }
    }
  }
}
```

Domain Message Builder

Implement a class that is responsible for building the domain model messages (`Sitecore.Social.Domain.Model.Message`) from the Field Editor fields on the client-side (Sitecore desktop). The Field Editor fields are used in the Social Connected message creation UI. You must implement one method:

- `Message BuildFrom(ICollection<FieldDescriptor> fields, string url, bool isContentPosting, Identifier accountIdentifier, bool postAutomatically)`. This method accepts information taken from items of messages in Sitecore (fields parameter) as well as additional parameters, such as:
  - `url` – the URL of the item that the relevant social message was created for.
  - `isContentPosting` – indicates whether this message was created for a content Sitecore item (if true), or for a goal item.
  - `accountIdentifier` – identifier of the account where the message will be posted.
  - `postAutomatically` – indicates whether a *Post Automatically* option was selected when creating a message.

The following code illustrates how to implement the `TumblrDomainMessageBuilder` class for Tumblr (*Sitecore.Social.Tumblr.Client* project):

```
namespace Sitecore.Social.Tumblr.Client.Builders
{
  using System.Collections.Generic;
  using System.Linq;
  using Sitecore.Data;
  using Sitecore.Social.Client.Builders;
  using Sitecore.Social.Domain.Model;
  using Sitecore.Social.Infrastructure.Extensions;
  using Sitecore.Social.Tumblr.Model;
  /// <summary>
  /// Tumblr domain message builder.
  /// </summary>
  public class TumblrDomainMessageBuilder : DomainMessageBuilder
  {
    /// <summary>
    /// Builds from.
    /// </summary>
    /// <param name="fields">The fields.</param>
    /// <param name="uri">The URI.</param>
    /// <param name="isContentPosting">if set to <c>true</c> [is content posting].</param>
    /// <param name="accountIdentifier">The account identifier.</param>
    /// <param name="postAutomatically">if set to <c>true</c> [post automatically].</param>
    /// <returns></returns>
    public override Message BuildFrom(ICollection<FieldDescriptor> fields, string uri, bool isContentPosting, Identifier accountIdenti
    {
      var tumblrMessageData = new TumblrMessageData();
      this.FillMessageDataFields(fields, tumblrMessageData);
      tumblrMessageData.Title = fields.Where(field => field.FieldID == Common.ItemIDs.TitleFieldId).Select(field => field.Value).Firs
      var tumblrNetworkIdentifier = Common.ItemIDs.TumblrNetworkItemId.GetIdentifier();
      return isContentPosting
```

```
        ? new Message(tumblrNetworkIdentifier, tumblrMessageData, this.GetPostConfiguration(fields, uri, accountIdentifier, postAutoma
        : new Message(tumblrNetworkIdentifier, tumblrMessageData, new GoalPostingConfiguration(uri));
    }
  }
}
```

Message Posting Provider

Implement a class that is responsible for preparing the data to be posted and calling the `NetworkProvider` class to post the message to a social network.

As a base class, use `Sitecore.Social.MessagePosting.Providers.MessagePostingProviderBase`.

You must override the `PostAll` method:

- In the `PostAll` method, call the `Post` method of the base `MessagePostingProviderBase` class and implement message posting for a specific account in the anonymous function.
- In the anonymous function, prepare the account data that you want to post and call the `PostMessage` method of the relevant network provider.

In the `PostAll` method, you can add your custom logic for creating a message for a social network, for example, replacing tokens.

The following code illustrates how to implement the `TumblrMessagePostingProvider` class for Tumblr (*Sitecore.Social.Tumblr* project). The `BuildTumblrMessage` method replaces the `$link` token in the message with the value of the *Link* field of the message:

```
namespace Sitecore.Social.Tumblr.MessagePosting.Providers
{
  using System;
  using System.Text;
  using Ninject;
  using Sitecore.Social.Infrastructure;
  using Sitecore.Social.Infrastructure.Utils;
  using Sitecore.Social.MessagePosting.Messages;
  using Sitecore.Social.MessagePosting.Providers;
  using Sitecore.Social.NetworkProviders;
  using Sitecore.Social.NetworkProviders.Interfaces;
  using Sitecore.Social.Tumblr.Networks.Messages;
  /// <summary>
  /// Tumblr message posting provider.
  /// </summary>
  public class TumblrMessagePostingProvider : MessagePostingProviderBase
  {
    /// <summary>
    /// The access token field name
    /// </summary>
    private const string AccessTokenFieldName = "AccessToken";
    /// <summary>
    /// The access token secret field name
    /// </summary>
    private const string AccessTokenSecretFieldName = "AccessTokenSecret";
    /// <summary>
    /// The message field name
    /// </summary>
    private const string MessageFieldName = "Message";
    /// <summary>
    /// The title field name
    /// </summary>
    ///
    private const string TitleFieldName = "Title";
    /// <summary>
    /// The link field name
    /// </summary>
    private const string LinkFieldName = "Link";
    /// <summary>
    /// Initializes a new instance of the <see cref="TumblrMessagePostingProvider"/> class.
    /// </summary>
    /// <param name="message">The message.</param>
```

```
public TumblrMessagePostingProvider(Message message)
  : base(message)
{
}
/// <summary>
/// Posts the messages in all accounts.
/// This method needs to be overridden in the descendant class and Post method of the current class needs to be called inside.
/// </summary>
public override void PostAll()
{
  this.Post(account =>
  {
    var networkProviderFactory = ExecutingContext.Current.IoC.Get<INetworkProviderFactory>();
    var tumblrMessagePostingProvider = networkProviderFactory.GetNetworkProvider<IMessagePosting>(account.Application, true);
    var networkProviderAccount = new Account()
    {
      AccessToken = account[AccessTokenFieldName],
      AccessTokenSecret = account[AccessTokenSecretFieldName],
      Application = account.Application
    };
    var tumblrPost = this.BuildTumblrPost(this.Message);
    var networkProviderMessage = new TumblrMessage()
    {
      Text = tumblrPost,
      Title = this.Message[TitleFieldName]
    };
    return tumblrMessagePostingProvider.PostMessage(networkProviderAccount, networkProviderMessage);
  });
}
/// <summary>
/// Builds the Tumblr post.
/// </summary>
/// <param name="message">The message.</param>
/// <returns></returns>
private string BuildTumblrPost(Message message)
{
  var messageText = new StringBuilder();
  if (!string.IsNullOrEmpty(message[LinkFieldName]))
  {
    var shortner = new GoogleUrlShortener();
    var link = message[LinkFieldName];
    var uri = shortner.ShortenUrl(new Uri(link));
    link = uri.AbsoluteUri;
    if (message[MessageFieldName].Contains("$link"))
    {
      messageText.Append(message[MessageFieldName].Replace("$link", link));
    }
    else
    {
      messageText.Append(message[MessageFieldName]);
      messageText.Append(" ");
      messageText.Append(link);
    }
  }
  else
  {
    messageText.Append(message[MessageFieldName].Replace("$link", string.Empty));
```

```
      }
      return messageText.ToString();
    }
  }
}
```

IMessagePosting interface

For the network to be displayed on social connected message posting UI, the network provider must implement the `Sitecore.Social.NetworkProviders.Interfaces.IMessagePosting` interface.

It contains one method:

- `PostResult PostMessage(Account account, Message message)` – responsible for posting a message to the relevant account. The method must return a `PostResult` instance, which encapsulates the result of message posting and contains two properties:
  - *MessageId* – the ID of the posted message
  - *Response* – additional response parameters, returned from the social network

The following code illustrates how to implement the `IMessagePosting` interface for Tumblr (*Sitecore.Social.Tumblr* project):

```
/// <summary>
    /// Posts the message.
    /// </summary>
    /// <param name="account">The account.</param>
    /// <param name="message">The message.</param>
    /// <returns></returns>
    public PostResult PostMessage(Account account, Message message)
    {
      var tumblrMessage = (TumblrMessage)message;
      var tumblrClient = this.GetTumblrClient(account);
      var textPost = PostData.CreateText(tumblrMessage.Text, tumblrMessage.Title);
      var postCreationInfo = tumblrClient.CreatePostAsync(this.GetAccountBasicData(account).FullName, textPost).Result;
      return new PostResult() { MessageId = postCreationInfo.PostId.ToString(CultureInfo.InvariantCulture) };
    }
    /// <summary>
    /// Gets the tumblr client.
    /// </summary>
    /// <param name="account">The account.</param>
    /// <returns></returns>
    private TumblrClient GetTumblrClient(Account account)
    {
      var factory = new TumblrClientFactory();
      return factory.Create<TumblrClient>(
        account.Application.ApplicationKey,
        account.Application.ApplicationSecret,
        new Token(account.AccessToken, account.AccessTokenSecret));
    }
```

Note

This sample code creates a text post. There are several other types of posts that can be created on Tumblr.

Message Renderer

Implement a message rendered class that inherits from the `Sitecore.Social.Client.MessagePosting.Renderers.IMessageRenderer` interface. It is responsible for rendering social network messages in the UI of the Social Connected module.

The following code illustrates how to implement the `TumblrMessageRenderer` class for Tumblr (*Sitecore.Social.Tumblr.Client* project):

```
namespace Sitecore.Social.Tumblr.Client.MessagePosting.Renderers
{
  using System.Globalization;
  using System.Text;
  using System.Web;
  using Ninject;
  using Sitecore.Social.Client.MessagePosting.Renderers;
  using Sitecore.Social.Client.MessagePosting.Renderers.Strategies;
  using Sitecore.Social.Domain.Model;
```

```
using Sitecore.Social.Infrastructure;
using Sitecore.Social.MessagePosting.Managers;
using Sitecore.Social.Tumblr.Model;
/// <summary>
/// Tumblr message renderer.
/// </summary>
public class TumblrMessageRenderer : IMessageRenderer
{
  /// <summary>
  /// Renders the message.
  /// </summary>
  /// <param name="message">The message.</param>
  /// <param name="postingConfigurationName">Name of the posting configuration.</param>
  /// <param name="messageTextRenderStrategy">The message text render strategy.</param>
  /// <returns></returns>
  public string RenderMessage(Message message, string postingConfigurationName, IMessageTextRenderStrategy messageTextRenderStrategy
  {
    var tumblrMessageData = message.MessageData as TumblrMessageData;
    var stringBuilder = new StringBuilder();
    if (tumblrMessageData != null)
    {
      var campaignId = tumblrMessageData.CampaignId.IsEmpty
                ? string.Empty
                : tumblrMessageData.CampaignId.StringValue;
      var link = string.Format(CultureInfo.CurrentCulture, @"<a href=""{0}"" target=""_blank"">{0}</a>", ExecutingContext.Current.Ioc
      var messageText = messageTextRenderStrategy.Render(message.Id, postingConfigurationName, HttpUtility.HtmlEncode(tumblrMessageDa
      stringBuilder.AppendFormat(
      @"<h2>{0}</h2>
      <p>{1}</p>",
      tumblrMessageData.Title,
      messageText);
      if (messageText.Contains("$link"))
      {
        stringBuilder = stringBuilder.Replace("$link", link);
      }
      else
      {
        if (!tumblrMessageData.Text.Contains("$link"))
        {
          stringBuilder.Append(link);
        }
      }
    }
    return stringBuilder.ToString();
  }
  /// <summary>
  /// Gets the message content CSS class.
  /// </summary>
  /// <returns></returns>
  public string GetMessageContentCssClass()
  {
    return string.Empty;
  }
 }
}
```

Update the social network configuration file

After you implement the new classes, you must update the social network configuration file to ensure that it refers to the new classes.

You must add\update the following elements:

Element	Attribute	Description
social\api\domainMessageBuilders\builder	networkName	Enter the name of the social network.
	builderType	Enter the full name of the domain message builder.
	mesageItemBase	Enter the full name of the message item class.
social\api\messageDataBuilders\builder	messageDataType	Enter the full name of the message data type.
	builderType	Enter the full name of the message data builder.
	type	Enter the full name of the message item class.
social\networks\network\items\message	MessageTemplateId	Enter the ID of the message template.
	Renderer	Enter the full name of the message renderer.
	rootCampaignItemId	Enter the ID of the social network message campaign item.
social\networks\network\messagePosting\campaigns	<campaign source=”ContentPosting”>	Enter the ID of the content posting campaign item.
	<campaign source=”GoalPosting”>	Enter the ID of the goal posting campaign item.
social\networks\network\messagePosting\publisher	type	Enter the full name of the message posting provider.

The following sample code shows the updated `Sitecore.Social.Tumblr.Config` file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <social>
      <api>
        <domainMessageBuilders>
          <builder networkName="Tumblr"                  builderType="Sitecore.Social.Tumblr.Client.Builders.TumblrDomainMessageBuild
        </domainMessageBuilders>
        <messageDataBuilders>
          <builder messageItemBaseType="Sitecore.Social.Tumblr.MessagePosting.Items.TumblrMessageItem, Sitecore.Social.Tumblr"
                   messageDataType="Sitecore.Social.Tumblr.Model.TumblrMessageData, Sitecore.Social.Tumblr"
                   builderType="Sitecore.Social.Tumblr.Builders.TumblrMessageDataBuilder, Sitecore.Social.Tumblr" />
        </messageDataBuilders>
      </api>
      <networks>
        <network name="Tumblr" ItemId="{ED602E87-21C4-432F-81F2-15F9E3A02DAF}" prefix="tb" icon="tumblr" url="https://www.tumblr.com">
          <items>
            <message type="Sitecore.Social.Tumblr.MessagePosting.Items.TumblrMessageItem, Sitecore.Social.Tumblr"
                     MessageTemplateId="{5A7B401B-FEE7-437C-9141-7134872C67FE}"
                     Renderer="Sitecore.Social.Tumblr.Client.MessagePosting.Renderers.TumblrMessageRenderer, Sitecore.Social.Tumblr.Cl
                     TextLimit="63000" />
          </items>
          <messagePosting>
            <campaigns rootCampaignItemId="{32D2CB8A-67DE-4F8F-9DF9-E3E9B9B17A90}">
              <campaign postingConfiguration="ContentPosting" itemId="{DC55C664-9305-48BF-BBF2-35155CCACB96}"/>
              <campaign postingConfiguration="GoalPosting" itemId="{C22F9FC2-E228-4223-89CE-64DCF6EFD8CC}"/>
            </campaigns>
```

```
          <publisher type="Sitecore.Social.Tumblr.MessagePosting.Providers.TumblrMessagePostingProvider, Sitecore.Social.Tumblr"/>
        </messagePosting>
        <providers>
          <provider type="Sitecore.Social.Tumblr.Networks.Providers.TumblrProvider, Sitecore.Social.Tumblr"/>
        </providers>
      </network>
    </networks>
  </social>
 </sitecore>
</configuration>
```


Add a new profile field to Social Connector

When a website visitor logs in to the website with their social network credentials, the module receives the visitor profile information from the social network. The module saves this information to a Sitecore user profile in the *extranet* domain and to a contact profile. The module saves the data from the social network to MongoDB.

The Social Connected module uses prefixes for social network fields to process user profile information. For example, `fb` is a prefix for Facebook fields.

If you installed the module, and a social network subsequently creates a new field for the user profile, you must add this field to Social Connector manually.

To add a new field to Social Connector:

1. Open the `[website_root]\app_config\include\social\sitecore.social.profilemapping.<network_name>.config` file.
2. Duplicate the block of the existing field configuration.
3. Set the attributes:
   - `Field enabled` – set this to *true* if you want Social Connector to get information from this field on a social network.
   - `sitecoreKey` – enter the name of the field as it appears in the Sitecore database.

   Set all the other attributes according to the official social network documentation.

Important

You should not change the sitecoreKey attribute values for the existing fields.

For more information about the user profile fields, see the official social network documentation:

- Facebook documentation.
- LinkedIn documentation.
- New Google+ API.
- Twitter documentation.

You can get more information about the `originalKey` attribute for Twitter in the properties of the `TwitterUser` class in the *TweetSharp* library.

You can get more information about the `originalKey` attribute for Google+ in the `WellKnownAttribute` class in the *DotNetOpenAuth* library.

Add share buttons to a webpage

You can add the Like (for Facebook), Tweet (for Twitter), +1 (for Google+), and LinkedIn Share (for LinkedIn) buttons on the website as Sitecore components. These buttons are called share buttons.

When you have added the share buttons to your webpage, you can assign campaigns and goals to them. When a website visitor clicks a share button, a specific goal or campaign is triggered and registered in Sitecore Analytics.

Before marketers can put share buttons on the website in the Experience Editor, Sitecore administrators must enable the controls for the appropriate placeholder:

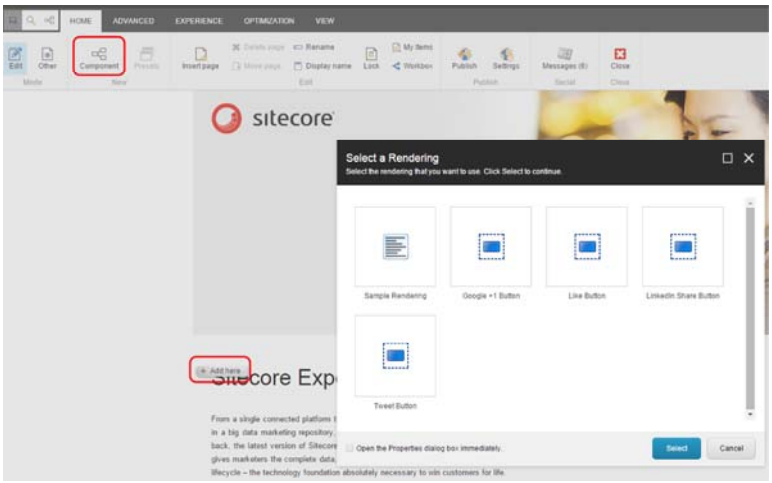
1. In the Content Editor, navigate to *sitecore/layout/placeholder settings* folder and select the placeholder item where you want to add the controls.
2. In the Allowed Controls field, add the share button controls to the list of selected controls.

   You can find the controls in the *sitecore/layout/sublayouts/social/sharing* folder. For the MVC layout, use the Social Connected renderings in the *sitecore/layout/Renderings/Social MVC/Sharing* folder

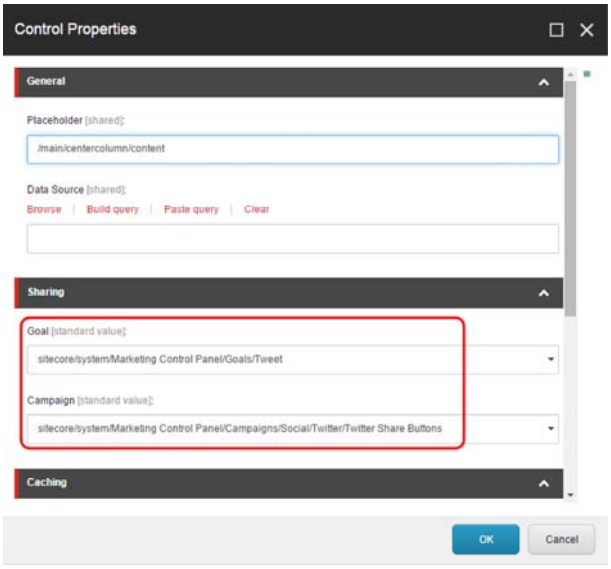
Note

For the custom Tweet button to work correctly, in the `website\layouts\system\social\sharing\TweetButton.ascx` file, make sure that the `data-url` and `data-counturl` attributes have the same value.

When you have configured placeholders, marketers can add the share buttons on a webpage in the Experience Editor as Sitecore components:



An appropriate goal and campaign is assigned to a share button by default:



Note

Changing the campaign assigned to the Tweet button resets the tweet counter to null.

Social Connected security roles

The following table lists the security roles for content authors and reviewers that you can use to configure content message posting:

Role	Description	Requires
sitecore\Author	Access to the Content Editor, Workbox, and Messages dialog box	Author (required) Reviewer (required)
sitecore\Sitecore Client Social Authoring	This role is a member of the *Sitecore\Author* role. Access to the Messages button on the Publish tab on the Experience Editor ribbon.	Author (required) Reviewer (required)
sitecore\Analytics Maintaining	Permissions to create goal/page event messages and campaigns for messages.	Author (optional)
sitecore\Analytics Reporting	Permissions to view Analytics reports for a message	Author (optional)
sitecore\Social Message Author	Permissions to create, edit, and remove all messages	Author (required)

sitecore\Facebook Message Reviewer	Permissions to edit Facebook messages	Reviewer (required)
sitecore\Twitter Message Reviewer	Permissions to edit Twitter messages	Reviewer (required)
sitecore\Social Marketer Message Reviewer	Permissions to edit social marketer messages	Reviewer (required)
sitecore\Social Message Workflow Editor	Permissions to use the Draft state and its commands of the *Social Message* workflow and the *Social Marketer Message* workflow	Author (required)
sitecore\Social Message Workflow Reviewer	Permissions to use the Awaiting Approval state and its commands of the *Social Message* workflow. Permissions to use the Awaiting Post Review, Post Accepted, and Removed states and their commands of the *Social Marketer Message* workflow	Reviewer (required)

Synchronize user information

When a visitor logs in to your website, Social Connected receives and saves the user profile information from the social network. The module saves social profile information to the Sitecore user profile in the *extranet* domain and to the contact profile.

The module receives the user profile information in two stages:

- When a website visitor enters their social network credentials, the module receives basic data from the social network: *ID, email, application key, application secret, access token*, and *access token secret*. The module uses this information to identify the visitor.
- When a visitor is identified, the module receives all the information that the visitor has allowed to share from the user profile on the social network. As a result, this information is received later than the basic data.

However, you can configure the module to receive all the user information simultaneously. Use the `Login` method of the `Sitecore.Social.Client.Common.Helpers.LoginHelper` class:

```
void Login(string networkName, bool updateSocialProfileAsync, Dictionary<string, object> additionalParameters, string callbackUrl = nu
```

where the `updateSocialProfileAsync` parameter is set to *False*.

By default, the module is configured to synchronize user information on login, once every two (or more) days. If a user logs in to your website one day after the last synchronization, the information is not synchronized, however, it is synchronized the next time the user logs in after two or more days have gone by.

You can configure Sitecore to synchronize user information every time the user logs in, or you can define the number of days after which you want it to synchronize.

Note

To update social profiles attached to a contact, the Synchronize Social Media Info agent locks the contact for the update, either during an online visit or by other processes. In this case, the Synchronize Social Media Info agent skips the social profile update until the contact is unlocked.

The lock period is specified in the `Social.Profile.ContactLeaseDuration` setting of the `Sitecore.Social.config` file. The default value of the lock period is 1000 milliseconds. If you are experiencing issues with updating social profiles due to lock-time expiration, increase the lock period.

This topic outlines how to:

- Synchronize user information every time a user logs in
- Synchronize user information after a specified number of days

Synchronize user information every time a user logs in

To get the latest information from the social network profile every time the user logs in to your website:

1. In the `website\app_config\include\social\sitecore.social.config` file, in the `settings` section, set the `Social.ProfileUpdating.Enabled` setting to *true*.
2. In the `Social.ProfileUpdating.DaysBeforeExpiration` setting, set the value to *0*.

Synchronize user information after a specified number of days

To get the latest information from the social network profile, you can set the value to a certain number of days regardless of when the user logs in:

1. In the `website\app_config\include\social\sitecore.social.config` file, in the `settings` section, set the `Social.ProfileUpdating.Enabled` setting to *true*.
2. In the `Social.ProfileUpdating.DaysBeforeExpiration` setting, set the value to the number of days after which the module synchronizes the user information with the social network. For example, if the value of this setting is *2*, the module synchronizes user information every two days when the scheduling agent runs.
3. In the `scheduling` section, in the `Sitecore.Social.Client.Tasks.SynchronizeSocialMediaInfo` agent, set the `interval` value:

```
<!-- The agent that synchronizes data with social networks. -->
<agent type="Sitecore.Social.Client.Tasks.SynchronizeSocialMediaInfo, Sitecore.Social.Client" method="Run" interval="00:00:00" />
```

Walkthrough: Adding a new social network to Social Connected

This walkthrough describes the required steps to add a new social network to work with Social Connected. The Tumblr social network is used as an example.

The walkthrough outlines how to:

- Prepare Sitecore for a new social network
- Create projects in Visual Studio
- Create a configuration file for the new social network
- Implement a network provider class
- Register the new social network in the configuration file
- Create icons in the media library
- Create the rendering parameters template
- Create the login control
- Implement the AuthGetCode and the AuthGetAccessToken methods
- Complete the authorization process

Note

For additional, optional functionality, you can also configure Social Messaging, configure Share buttons using Sitecore MVC or configure Share buttons using Sitecore sublayouts.

Prepare Sitecore for a new social network

To prepare Sitecore for a new social network:

1. Create icons for a new social network. At a minimum, you must create a set of two icons with dimensions of 16x16 and 32x32 pixels. These icons are rendered in the module and the Sitecore UI. The icons must have the same name. Put the icons in the following folders:
   - The `Website\sitecore\shell\themes\standard\custom\16x16` folder.
   - The `Website\sitecore\shell\themes\standard\custom\32x32` folder.
2. Create a social network item. In the Sitecore Content Editor, navigate to the *Networks* folder (`sitecore/system/social/networks`) and create an item based on the *Network* template (`sitecore/templates/system/social/network`). Assign an icon to the created item using the icon with 16x16 pixels.
3. Create a social network web application to connect Social Connected with the social network. For example, to create a Tumblr web application:
   - Go to https://www.tumblr.com/oauth/apps and create an application.
   - After you create the application, go to https://www.tumblr.com/oauth/apps, get and save the OAuth Consumer Key and the Secret Key for the appropriate application. Use these values when you create a Sitecore item for the web application.
4. In the Content Editor, navigate to the *Default* folder, (`sitecore/system/social/applications/default`) and create an item based on the *Application* template (`sitecore/templates/system/social/application`).
5. In the new item, enter the following information:

Field	Value
`ApplicationId`	The OAuth Consumer Key of the web application.
`ApplicationSecret`	The Secret Key of the web application.
`Network`	Select the social network from the list, for example, Tumblr.

Create projects in Visual Studio

After you have prepared Sitecore to add a new social network, you must create the relevant projects in Visual Studio. The default project layout for a social network in the Social Connected module usually contains three projects:

- `Sitecore.Social.<NetworkName>` – contains network-related logic (network provider, posting provider, etc.).
- `Sitecore.Social.<NetworkName>.Client` – contains ASP.NET web forms client-side related logic and controls (login button control, share button control, etc.).
- `Sitecore.Social.<NetworkName>.Client.mvc` – contains ASP.NET MVC client-side related logic and controls (controllers, share button view, etc.).

To create projects in Visual Studio:

1. In Visual Studio, create a new Class Library project. Name it, for example, *Sitecore.Social.Tumblr*. Add references to the following assemblies:
   - `Sitecore.Kernel.dll`
   - `Sitecore.Social.dll`
   - `Sitecore.Social.Domain.dll`
   - `Sitecore.Social.Infrastructure.dll`
   - `Sitecore.Social.NetworkProviders.dll`
   - `System.Web.dll`
   - `Ninject.dll`
   - `DontPanic.TumblrSharp.dll`
   - `DontPanic.TumblrSharp.Client.dll`
   - `DontPanic.TumblrSharp.Net45.dll`
2. Set the following project properties:
   - Configuration – *Release*
3. For the client-side controls, create a Class Library project, for example *Sitecore.Social.Tumblr.Client*.
4. Add references to the following assemblies:
   - `Sitecore.Kernel.dll`
   - `Sitecore.Social.dll`
   - `Sitecore.Social.Domain.dll`
   - `Sitecore.Social.Client.dll`
   - `Sitecore.Social.Client.Common.dll`
   - `Sitecore.Social.Infrastructure.dll`
   - `Sitecore.Social.Tumblr.dll`
   - `System.Web.dll`

    - `Ninject.dll`
5. Set the following project properties:
    - Configuration – *Release*
    - Output path – the path to the `website\bin` folder of your Sitecore website
6. For client-side MVC controls, create a Class Library project, for example *Sitecore.Social.Tumblr.Client.Mvc*.
7. Add references to the following assemblies:
    - `Sitecore.Kernel.dll`
    - `Sitecore.Social.Client.Mvc.dll`
    - `Sitecore.Social.Client.Common.dll`
    - `Sitecore.Social.Tumblr.dll`
    - `System.Web.Mvc.dll`
    - `System.Web.dll`
8. Set the following project properties:
    - Configuration – *Release*
    - Output path – path to the `website\bin` folder of your Sitecore website

Create a configuration file for the new social network

After you have created the projects in Visual Studio, you must create a configuration file for the social network.

To create a configuration file:

1. Create the file and name it, for example, `Sitecore.Social.Tumblr.config`.
2. Navigate to `Website\App_Config\Include\Social` and place this file in the *Social* folder.

For convenience, you should add a reference to this configuration file in your Visual Studio project (for example, in the *Sitecore.Social.Tumblr* project).

Implement a network provider class

After you create a configuration file for the social network, you must implement a network provider class.

To implement a network provider class:

1. Create a network provider class that is inherited from the `Sitecore.Social. NetworkProviders.NetworkProvider` class.
2. Name this class, for example, `Sitecore.Social.Tumblr.Networks.Providers.TumblrProvider`.

Register the new social network in the configuration file

After you have prepared Sitecore and created projects in Visual Studio, you must register the new social network in the configuration file to let the module find the social network settings and create its provider.

To register a social network in the configuration file, for example, the `Sitecore.Social.Tumblr.config` file:

1. Open the `Sitecore.Social.Tumblr.config` file.
2. In `configuration/Sitecore/social/networks` add the `<network>` and `<provider>` nodes:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
<sitecore>
<social>
<networks>
<network name="Tumblr" ItemId="{ED602E87-21C4-432F-81F2-15F9E3A02DAF}" prefix="tb" icon="tumblr" url="https://www.tumblr.com">
<providers>
<provider type="Sitecore.Social.Tumblr.Networks.Providers.TumblrProvider, Sitecore.Social.Tumblr"/>
</providers>
</network>
</networks>
</social>
</sitecore>
</configuration>
```

   Use the following required attributes of the `<network>` node:

    - `Name` – network name.
    - `ItemID` – ID of the social network item in Sitecore that you created.
    - `Prefix` – network prefix.
    - `Icon` – social network icon that you created.
    - `URL` – URL of the network.
    - In the type attribute of the `<provider>` node, specify the class of the social network that you created, for example, `Sitecore.Social.Tumblr.Networks.Providers.TumblrProvider`.

After you have registered the new social network in the configuration file, you must set up the Social Connector feature by preparing the connector controls so that a user can log in to your website using the new social network.

Create icons in the media library

The controls in Social Connected use images from the media library as icons for the login buttons. To create an icon for the Social Connected login controls:

1. Navigate to `/sitecore/media library/Images/Social/Connector` and in the *Connector* folder, upload an icon image.
2. Fill in the following fields:

- Alt: The name of the new social network, for example, *Tumblr*
- Width: *24*
- Height: *24*

Create the rendering parameters template

Social Connected uses Sitecore rendering parameters for passing the configuration options to the API, such as:

- `Application`
- `Callback URL`

To create the rendering parameters template:

1. Navigate to `/sitecore/templates/System/Social/Connector` and in the *Connector* folder create a new template. Name it, for example, `Login with Tumblr Button Parameters`. Alternatively, you can duplicate an existing template.
2. In the Base Template field, enter `/sitecore/templates/System/Social/Connector/Base Login Button Parameters`.
3. In the new template, on the Builder tab, add the Application field with a `Droplink` field type.
4. Select the Shared check box for this field and, in the Source field, enter the following query using the ID of the network item:
   `query:/sitecore/system/social/applications//*[@Network='<ID of the network item>']`

Create the login control

You can use either Sitecore sublayouts or Sitecore MVC renderings to create the login control, depending on the functionality you use on your website.

Using sublayouts

To create the login control sublayout:

1. In the Content Editor, navigate to `/sitecore/layout/Sublayouts/Social/Connector`.
2. In the *Connector* folder, click Sublayout to create a new sublayout item
3. In the wizard, specify the following parameters:

Parameter	Name
Name	Enter the name for your sublayout item, for example, *Login with Tumblr*
Item Location	`/sitecore/layout/Sublayouts/Social/Connector`
File Location	`website/layouts/system/social/connector`

4. Select the Create Associated C# Code Files check box. Alternatively, you can create the code files manually later.
5. In the Parameters Template field, select the new template that you created earlier, for example, the *Login with Tumblr Button Parameters* template.
6. Add references to the new sublayout files to the relevant Visual Studio project, for example, the Sitecore.Social.Tumblr.Client project.
7. Add the following code to the control's `.ascx` file:

```
<%@ Control Language="C#" AutoEventWireup="true"    Inherits="Sitecore.Social.Tumblr.Client.Connector.Controls.LoginWithTumblr"
  TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<asp:ImageButton runat="server" ID="tumblrLoginButton" CssClass="button" ToolTip="Login with Tumblr" OnClick="TumblrLoginButtonOn
```

8. Use the `Sitecore.Social.Client.Connector.Controls.LoginButtonBase` class as a base class for your user control. Call its `Login` method in the `OnClick` handler of your login button and pass the network name as a parameter, for example, for `Tumblr`:

```
namespace Sitecore.Social.Tumblr.Client.Connector.Controls
{
  using System;
  using System.Web.UI;
  using System.Web.UI.WebControls;
  using Sitecore.Globalization;
  using Sitecore.Social.Client.Connector.Controls;
  /// <summary>
  /// Login with Tumblr connector control.
  /// </summary>
  public class LoginWithTumblr : LoginButtonBase
  {
    /// <summary>
    /// The Tumblr login button
    /// </summary>
    protected ImageButton tumblrLoginButton;
    /// <summary>
    /// Handles the Load event of the Page control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="EventArgs"/> instance containing the event data.</param>
```

```
        protected void Page_Load(object sender, EventArgs e)
        {
          if (this.Page.IsPostBack)
          {
            return;
          }
          this.tumblrLoginButton.ImageUrl = this.GetLoginButtonImageUrl(Common.ItemIDs.LoginWithTumblrImageId);
          this.tumblrLoginButton.ToolTip = Translate.Text(Sitecore.Context.User.IsAuthenticated ? Common.Texts.AttachTumblrAccount : (
        }
        /// <summary>
        /// Processing pressing of tumblrLoginButton button.
        /// </summary>
        /// <param name="sender">
        /// The sender.
        /// </param>
        /// <param name="e">
        /// The <see cref="System.Web.UI.ImageClickEventArgs"/> instance containing the event data.
        /// </param>
        protected void TumblrLoginButtonOneClick(object sender, ImageClickEventArgs e)
        {
          this.Login("Tumblr");
        }
      }
    }
```

The base implementation of the `Login` method does the following:

- Processes the `Callback URL` rendering parameter.
- Calls the `Sitecore.Social.Client.Common.Helpers.ILoginHelper.Login` method.

The `ILoginHelper.Login` method first populates rendering parameters and makes sure that they are passed along the authentication flow. Then, depending on whether the user is authenticated, the method calls the appropriate method of the `ConnectorClientManager` class.

If the current user is not authenticated, the `LogOnUser` method is called. Otherwise, the social profile of the current network will be attached to the currently authenticated user using the `AttachToUser` method.

The `GetLoginButtonImageUrl` method accepts an ID of an image in the Sitecore media library and returns its URL. It is used to set an image on the button. Use the icon image that you have just created.

Best practice

The `Common.ItemIDs` class contains identifiers of items. It is best practice in Sitecore development to store well-known IDs in a separate class. For example, the *Sitecore.Social.Tumblr* project:

```
namespace Sitecore.Social.Tumblr.Common
{
  using Sitecore.Data;
  /// <summary>
  /// The Sitecore item identifiers.
  /// </summary>
  public static class ItemIDs
  {
    /// <summary>
    /// The Title field identifier
    /// </summary>
    public static readonly ID TitleFieldId = new ID("{25563C1A-D591-422E-A04E-138B98F06E78}");
    /// <summary>
    /// The Tumblr network item identifier
    /// </summary>
    public static readonly ID TumblrNetworkItemId = new ID("{ED602E87-21C4-432F-81F2-15F9E3A02DAF}");
    /// <summary>
    /// The login with Tumblr image identifier
    /// </summary>
    public static readonly ID LoginWithTumblrImageId = new ID("{309D3797-5130-4707-B157-A316F9C805CF}");
  }
```

```
}
```

The code below refers to the `Common.Texts` class. It is best practice in Sitecore development to store translation keys a in a separate class. For example:

```
namespace Sitecore.Social.Tumblr.Common
{
  /// <summary>
  /// Text constants
  /// </summary>
  public static class Texts
  {
    public const string LoginWithTumblr = "Login with Tumblr";
    public const string AttachTumblrAccount = "Attach Tumblr account";
    public const string ShareOnTumblr = "Share on Tumblr";
  }
}
```

Using Sitecore MVC renderings

Social Connected supports the controller renderings feature in Sitecore.

To create a login control using Sitecore MVC renderings:

1. In Visual Studio, create a controller in the new Class Library project, for example, the *Sitecore.Social.Tumblr.Client.Mvc* project. Use the `Sitecore.Social.Client.Mvc.Areas.Social.Controllers.ConnectorController` as a base class:

   ```
   namespace Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Controllers
   {
     using System.Web.Mvc;
     using Sitecore.Globalization;
     using Sitecore.Social.Client.Mvc.Areas.Social.Controllers;
     /// <summary>
     /// Represents Tumblr connector controller.
     /// </summary>
     public class TumblrConnectorController : ConnectorController
     {
       /// <summary>
       /// Initializes a new instance of the <see cref="TumblrConnectorController"/> class.
       /// </summary>
       public TumblrConnectorController()
         : base("Tumblr", Common.ItemIDs.LoginWithTumblrImageId)
       {
       }
       /// <summary>
       /// Gets the Tumblr login view.
       /// </summary>
       /// <returns>The login view.</returns>
       public ActionResult Index()
       {
         string tooltip = Translate.Text(Context.User.IsAuthenticated ? Common.Texts.AttachTumblrAccount : Common.Texts.LoginWithTuml
         return this.LoginPartialView(tooltip);
       }
     }
   }
   ```

   The `LoginPartialView` method accepts a parameter – `tooltip`, the tooltip for the login button.

   After a user clicks the button, the system performs a request to the `Login` action of `ConnectorController`. It internally calls the `ILoginHelper.Login` method.

   The constructor of the base class accepts two parameters:

   - `networkName` – the name of the network
   - `networkIconItemID` – the ID of an image in the Sitecore media library, which represents the button icon. Use the image you just created.
2. In Sitecore, navigate to `/sitecore/layout/Renderings/Social MVC/Connector/`.
3. In the *Connector* folder, to create a new MVC controller rendering, click Controller rendering and enter a name for the new item.
4. In the Controller field, enter: `Sitecore.Social.Tumblr.Client.Mvc.Areas.Social.Controllers.TumblrConnectorController, Sitecore.Social.Tumblr.Client.Mvc.`
5. In the Parameters Template field, enter: `/sitecore/templates/System/Social/Connector/Login with Tumblr Button Parameters template.`

After you have prepared the connector controls, you must configure your solution to receive user credentials. To do this, you must implement the `AuthGetCode` method and the `AuthGetAccessToken` method.

Implement the AuthGetCode and the AuthGetAccessToken methods

The `AuthGetCode` method must compose an authorization URL and redirect users to the social network. The `AuthGetAccessToken` method must get the *OAuth* access token from the social network.

In your implementation of the `AuthGetCode` method, do the following:

1. Get the request token passing the `oauth_callback` parameter.

   The `oauth_callback` parameter is an URL that redirects the user from the social network to complete the authentication process. The format of the `oauth_callback` parameter is: http://yourWebSite/layouts/system/Social/Connector/SocialLogin.ashx?type=access&state=authArgsState.

   The query string parameters of the `oauth_callback` parameter are:

   - `type=access` – this parameter makes `SocialLogin.ashx` call the `AuthGetAccessToken` method.
   - `state=authArgsState` – places your `authArgs.StateKey` in this parameter to restore it after the user is returned from the social network.
2. Compose an authorization URL using the request token and redirect the user.

For example, in this sample code the `AuthGetCode` method has been implemented for Tumblr using the Tumblr# library:

```
/// <summary>
    /// Gets the code.
    /// </summary>
    /// <param name="args">The arguments.</param>
    public void AuthGetCode(AuthArgs args)
    {
      var callbackUrl = WebUtil.GetFullUrl(string.Format("{0}?type=access&state={1}", Paths.SocialLoginHandlerPath, args.StateKey));
      var clientFactory = new OAuthClientFactory();
      var oauthClient = clientFactory.Create(args.Application.ApplicationKey, args.Application.ApplicationSecret);
      var requestToken = oauthClient.GetRequestTokenAsync(callbackUrl).Result;
      args.InternalData[RequestTokenKey] = requestToken;
      RedirectUtil.Redirect(oauthClient.GetAuthorizeUrl(requestToken).ToString());
    }
```

The `AuthGetAccessToken` method must get the *OAuth* access token from the social network and then call the `InvokeAuthCompleted` method of the base class passing the initialized instance of `AuthCompletedArgs`.

The following sample code illustrates how to implement the `AuthGetAccessToken` method for Tumblr:

```
/// <summary>
    /// Gets an access token.
    /// </summary>
    /// <param name="args">The arguments.</param>
    public void AuthGetAccessToken(AuthArgs args)
    {
      var oauthClientFactory = new OAuthClientFactory();
      var oauthClient = oauthClientFactory.Create(args.Application.ApplicationKey, args.Application.ApplicationSecret);
      var accessToken = oauthClient.GetAccessTokenAsync((Token)args.InternalData[RequestTokenKey], HttpContext.Current.Request.Url.ToS
      if (args.CallbackType == null)
      {
        return;
      }
      var authCompletedArgs = new AuthCompletedArgs
      {
        Application = args.Application,
        AccessToken = accessToken.Key,
        AccessTokenSecret = accessToken.Secret,
        CallbackPage = args.CallbackUrl,
        ExternalData = args.ExternalData,
        AttachAccountToLoggedInUser = args.AttachAccountToLoggedInUser,
        IsAsyncProfileUpdate = args.IsAsyncProfileUpdate
      };
      this.InvokeAuthCompleted(args.CallbackType, authCompletedArgs);
    }
```

After you have configured your solution to receive user credentials, you must complete the authorization process to finish configuring Social Connector.

Complete the authorization process

To complete the authorization process and finish configuring Social Connector:

1. Inherit the `IGetAccountInfo` interface for the network provider that you have just created.
2. Implement the `GetAccountBasicData` method of the `IGetAccountInfo` interface:

   Note

   Some social networks, for example, Tumblr, do not provide a user identifier, so `userInfo.Name` is used as an ID. The user name corresponds to the name of the primary blog, and it is unique.

```
/// <summary>
    /// Gets the account basic data.
    /// </summary>
    /// <param name="account">The account.</param>
    /// <returns></returns>
    public AccountBasicData GetAccountBasicData(Account account)
    {
      var userInfo = this.GetUserInfo(account);
      return new AccountBasicData
      {
        Account = account,
        FullName = userInfo.Name,
        Id = userInfo.Name
      };
    }
```

   The `GetUserInfo` method:

```
/// <summary>
    /// Gets the user information.
    /// </summary>
    /// <param name="account">The account.</param>
    /// <returns></returns>
    private UserInfo GetUserInfo(Account account)
    {
      var tumblrClient = this.GetTumblrClient(account);
      return tumblrClient.GetUserInfoAsync().Result;
    }
```

   The `GetTumblrClient` method:

```
/// <summary>
    /// Gets the tumblr client.
    /// </summary>
    /// <param name="account">The account.</param>
    /// <returns></returns>
    private TumblrClient GetTumblrClient(Account account)
    {
      var factory = new TumblrClientFactory();
      return factory.Create<TumblrClient>(
        account.Application.ApplicationKey,
        account.Application.ApplicationSecret,
        new Token(account.AccessToken, account.AccessTokenSecret));
    }
```

3. Implement the `GetAccountInfo` method to receive the user data from the social network and enable the social profile updating functionality for the social network.

   The `GetAccountInfo` method contains the following parameters:

   - *Account* – a social network account with the user credentials that include the access token and the access token secret.
   - *IEnumerable<FieldInfo>* – the collection of fields to request from the social network. You must declare these fields in the `Sitecore.Social.ProfileMapping.YourSocialNetwork.config` file.

   Sample code:

```
/// <summary>
    /// Gets the account information.
```

```
        /// </summary>
        /// <param name="account">The account.</param>
        /// <param name="acceptedFields">The accepted fields.</param>
        /// <returns></returns>
        public IEnumerable<Field> GetAccountInfo(Account account, IEnumerable<FieldInfo> acceptedFields)
        {
          Assert.IsNotNull(acceptedFields, "AcceptedFields collection must be filled");
          var acceptedFieldsList = acceptedFields.ToList();
          if (!acceptedFieldsList.Any())
          {
            yield break;
          }
          var userInfo = this.GetUserInfo(account);
          foreach (var acceptedField in acceptedFieldsList.Where(acceptedField => !string.IsNullOrEmpty(acceptedField.OriginalKey)))
          {
            var propertyInfo = ReflectionUtil.GetPropertyInfo(userInfo, acceptedField.OriginalKey);
            if (propertyInfo == null)
            {
              ExecutingContext.Current.IoC.Get<ILogManager>().LogMessage(
                string.Format(CultureInfo.CurrentCulture, "There is no property \"{0}\" in a UserInfo object", acceptedField.OriginalK
                LogLevel.Warn,
                this);
              continue;
            }
            var propertyValue = propertyInfo.GetValue(userInfo);
            if (propertyValue == null)
            {
              continue;
            }
            yield return new Field
            {
              Name = this.GetFieldSitecoreKey(acceptedField),
              Value = propertyValue.ToString()
            };
          }
        }
```

4. To map the information in the social network user profile to the social profile fields, navigate to `Website\App_Config\Include\Social` and in the *Social* folder, create a profile mapping configuration file. Name it, for example, `Sitecore.Social.ProfileMapping.Tumblr.config`.
5. In the configuration file, create a `sitecore/social/profileKeyMappings/network` section with the name attribute.
6. In the network section, add some field sections that contain the following required attributes:
   - `enabled` – receiving information from the network for this field. Possible values: *true* or *false*.
   - `originalKey` – the field name in the social network. The sample implementation of the `GetAccountInfo` method expects that the `originalKey` attribute stores the name of the property of the `UserInfo` class (from Tumblr#).
   - `sitecoreKey` – the field name in the social profile.
   - `text` – the display name of the field that is rendered in the module UI.

The field section contains optional attributes. The module receives the field settings as input parameters of the `GetAccountInfo` method. You can use the attributes of the field section to create requests to the social network and then parse the response to receive the field values. Complex responses require extra data. You can use some optional field attributes to configure the extra data.

Sample mapping:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <social>
      <profileKeyMappings>
        <network name="Tumblr">

          <field enabled="true" originalKey="name" sitecoreKey="tb_Name" text="Name" />
          <field enabled="true" originalKey="LikesCount" sitecoreKey="tb_LikesCount" text="Likes Count" />
          <field enabled="true" originalKey="FollowingCount" sitecoreKey="tb_FollowingCount" text="Following Count" />
        </network>
      </profileKeyMappings>
```

```
    </social>
  </sitecore>
</configuration>
```


The Social Connected events

The Social Connected module events let you process activities related to Social Connector and to messages. The `Sitecore.Social.config` file contains the events; the `Sitecore.Social.ScalabilitySettings.config` file contains the remote events.

Event	Description
social:connector:user:created	A new Sitecore user has been created.
social:connector:user:created:remote	A new Sitecore user has been created on a remote Sitecore instance.
social:connector:user:loggedin	A user has logged into the website using social network credentials.
social:connector:user:loggedin:remote	A user has logged into the website on a remote Sitecore instance using social network credentials.
social:connector:user:socialprofileattached	A user has attached a new social profile to their current profile.
social:connector:user:socialprofileattached:remote	A user has attached a new social profile to their current profile on a remote Sitecore instance.
social:message:created	A social message has been created.
social:message:created:remote	A social message has been created on a remote instance.
social:message:deleted	A social message has been deleted.
social:message:deleted:remote	A social message has been deleted on a remote instance.
social:message:updated	A social message has been updated.
social:message:updated:remote	A social message has been updated on a remote server.

To disable a remote event, comment the handler that calls this event in the `Sitecore.Social.ScalabilitySettings.config` file. For example, to disable the `social:message:created:remote` event, in the `Sitecore.Social.ScalabilitySettings.config` file, comment this code:

```
<event name="social:message:created">
<!-- Raises the "social:message:created:remote" event. -->
<!--<handler type="Sitecore.Social.Client.MessagePosting.Handlers.SocialMessageCreatedHandler, Sitecore.Social.Client" method="OnSocia
      </event>
```


Create a goal message

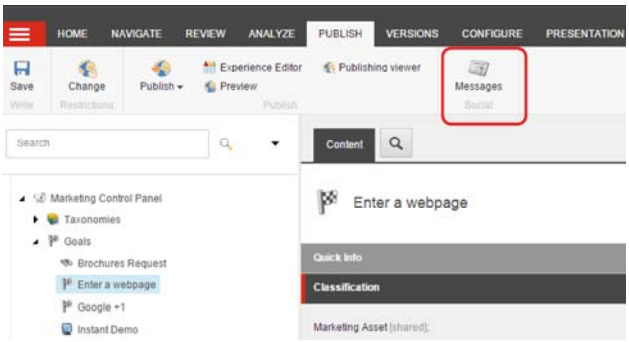
In Social Connected, goal messages are messages that are assigned to a goal. You can automatically post messages to a visitor's timeline in the social network every time the visitor triggers a goal on your website.

Before creating a goal message, make sure that your website:

- Has the Social Connector configured.
- Has the necessary log-in controls.

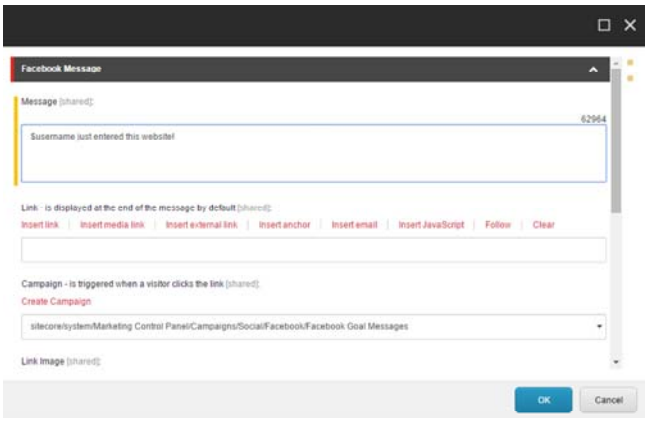
To create a goal message and make it ready to be automatically posted once a logged in visitor triggers a goal on your website:

1. In the Marketing Control Panel, create the goal that you want your visitors to trigger or use an existing one.
2. Select a goal item, and then on the Publish tab, click Messages.



3. In the Messages dialog box, click Create a Facebook message or Create a Twitter message.

   You can create several messages for one goal.



   You can use a token to personalize your message. A token is replaced with the corresponding value when the message is posted. You can add a token in this format: `$<token_name>`. Contact your Sitecore administrator to configure tokens before using them.

4. In the Message field, enter the text of the message.
5. In the Link field, enter the link to the webpage that you want the website visitor to go to when they have clicked the message in the social network. By default, the link refers to the current page.

   Note

   For a Facebook message, to style the link, use the Link Image, Link Title, and Link Description fields.

6. In the Campaign field, select an existing campaign or create a new one that is triggered when a visitor clicks the link in the message.

   If you assign a campaign to the message, you can review message statistics in Sitecore Analytics. For example, you can see statistics for a value, visits, and value per visits for all the visitors who click a link in this message. The campaign must be deployed before you can use it in the message.

7. Deploy the goal and then publish the goal item.
8. Make sure that the goal is ready to be triggered by the visitor. For example, assign a goal to an item and publish the item or add share buttons with the assigned goals to the webpage.
9. Using the Incremental publishing option, publish the `sitecore/Social/Messages` folder.

Now your goal message is configured and it is posted to the visitor's timeline if:

- The visitor has an appropriate social network profile attached to their contact.

  For example, if you have created a Twitter message on a goal, this message is posted if the visitor has Twitter attached to their contact.

- The visitor triggers the appropriate goal.

Create and post a content message

In the Social Connected module there are four types of messages:

- Content messages

  The messages that are assigned to a Sitecore item. You can post messages about Sitecore content automatically to the timelines of the selected social network accounts when you publish Sitecore items. You can post messages manually without publishing Sitecore items. You can also create social marketer messages to post messages to any social network.

- Goal messages

  The messages that are assigned to a goal. You can automatically post messages to the visitor's timeline in the social network every time they trigger a goal on the website.

- Detached messages

The messages that are not assigned to either a Sitecore item or a goal. You can post detached messages using the API.

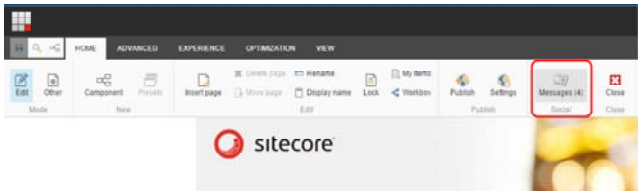
- Direct messages

  The messages that are not assigned to either a Sitecore item or a goal. You can create and post direct messages using the API. In contrast to a detached message, a direct message is not stored in Sitecore, and you cannot get its Sitecore Analytics statistics. You can only retrieve a list of the comments to this message in Facebook using the API.

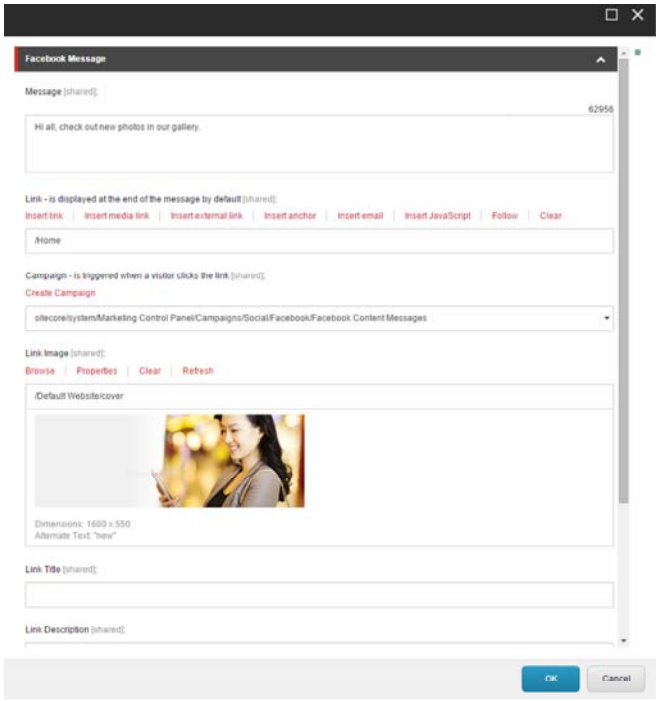
You can create and post content messages to Facebook and Twitter either immediately or automatically along with the publishing of the corresponding Sitecore item. You can also create a social marketer message to send a draft of your message to the marketer.

To create a content or social marketer message:

1. In the Experience Editor, open the page that you want to bind a message to.
2. On the ribbon, on the Home tab, in the Social section, click Messages.



3. In the Messages dialog box, click Create a Facebook message, Create a Twitter message, or Create a Social Marketer message.
4. In the dialog box for the message, in the Message field, enter the text of the message.



5. Enter the additional fields to specify the settings for your message:
   - For a social marketer message, you can enter a comment to a media marketer in the Comments field.
   - In the Link field, enter the link to the webpage that you want the website visitor to go to when they have clicked the message in the social network. By default, the link refers to the current page.
   - In the Campaign field, select an existing campaign or create a new one that is triggered when a visitor clicks the link in the message.

     If you assign a campaign to the message, you can review message statistics in Sitecore Analytics, for example, a value, visits, and value per visits for all the visitors who click on a link in this message. The campaign must be deployed before you can use it in the message.

   - For a Facebook message, to style the link, use the Link Image, Link Title, and Link Description fields.
6. In the Account field, select one or more social network or social marketer accounts that you want to post the message to. If you select several accounts, several messages are created – one message for each selected account.
7. To post the Facebook or Twitter message automatically when you publish the current Sitecore item, click Post automatically.

   If the message is posted with a delay after the Sitecore item is published, contact your Sitecore administrator to set a shorter posting interval.

   Note

   If for the selected account, messages are placed in a workflow, only the messages in the final workflow state are posted to a social network.

8. Click OK.
9. To post the message, in the Message dialog box, click Post now. Do not click Post now for the messages that are posted automatically.

   You can immediately post a message that is configured to be posted automatically when the item is published. In this case, the message is not posted second time when the item is published.

A message must meet all of the following requirements to be automatically posted to a social network:

- In the message configuration window, the Post automatically check box is selected.

- The message is either in the final state of a workflow or not in a workflow.
- The message has never been posted before.
- The corresponding content item is published after the message was created.

In a Twitter message, the link is inserted at the end of the message by default. To put the link in the middle of the message text, in the Twitter message dialog box, in the Text field, insert the `$link` token and in the Link field, enter the URL. Facebook does not work with this token.

You must use Workbox to edit and review the messages in a workflow.

Get a visitor's approval to post goal messages

When you have configured the Social Connected module to post goal messages to the social network when a visitor achieves a goal on the website, this functionality applies to all visitors.

If you want to ask website visitors whether they want messages to be posted on their social network timelines, you must disable the default sending of messages and create a UI control to ask visitors what they prefer:

1. In the `website\app_config\include\social\sitecore.social.config` file, set the value of the Social.AllowMessageGoalPostingByDefault setting to *false*.
2. Use the Sitecore API to create a UI control that asks the visitor whether they allow the Social Connected module to post goal messages to their social network timelines.

   Use the following methods of the `Sitecore.Social.Api.PostingPreferenceManager` class:

   - `GetPostingPreference`
   - `SetPostingPreference`

   For example, the following code snippet asks the current visitor whether they will allow the posting of a specific message. The `messageId` variable contains the message identifier:

```
var postingPreferenceManager = new PostingPreferenceManager();
var status = postingPreferenceManager.GetPostingPreference(Sitecore.Context.User.Name, messageId).AutomaticPostingPreference;
var newStatus = status == AutomaticPostingPreference.Allowed ? AutomaticPostingPreference.Denied : AutomaticPostingPreference.All
      postingPreferenceManager.SetPostingPreference(Sitecore.Context.User.Name, messageId, newStatus);
```


Social Connected messages

In the Social Connected module there are the following types of messages:

- Content Messages – messages that are assigned to a Sitecore item. You can post messages about Sitecore content automatically to the timelines of the selected social network accounts when you publish Sitecore items. You can post messages manually without publishing Sitecore items or you can create social marketer messages to post to any social network.
- Goal Messages – messages that are assigned to a goal. You can automatically post messages to the visitor's timeline in the social network every time they trigger a goal on your website.
- Detached messages – messages that are not assigned to either a Sitecore item or a goal. You can create and post detached messages using the API. When the detached message is posted, you can get its Sitecore Analytics statistics.
- Direct messages – messages that are not assigned to either a Sitecore item or a goal. You can create and post direct messages using the API. In contrast to a detached message, a direct message is not stored in Sitecore, and you cannot get its Sitecore Analytics statistics. You can only retrieve a list of the comments to this message in Facebook using the API.

Content messages

You can use the Social Connected module to bind messages to content items and post these messages automatically to social networks when you publish your Sitecore items. You can also post the messages manually without publishing the content items. These messages can also contain the link to webpages, so that your visitors can go directly from the social network to your website. You can assign a campaign to this link, and track statistics on the campaign in Analytics.

When you have a new campaign, for example, with the launch of a new product, you can create a new webpage and announce it on the social networks. To attract more attention to the new product, you can automatically post messages to the social network accounts letting your contacts know that you have unveiled a new product.

This posting feature works with Facebook and Twitter. You can post messages to both a visitor's timeline in Facebook and to a Facebook page. You can post messages (tweets) to the timeline of a Twitter user who has used the web application to create a Twitter account in Sitecore.

Social marketer messages

With social marketer messages, you can post messages to any social network. A social marketer message is a request to a media marketer. In the request, a content author informs the social media marketer that new content is ready to be promoted in social networks. The content author provides a draft of the message, comments with additional information, a link to the webpage, and the relevant campaign. A social media marketer creates and promotes your content messages in any social network.

Media marketers must have access to Sitecore Workbox to be able to work with social marketer messages.

Posting and reviewing

You can use the standard Sitecore workflow functionality to make sure the messages are reviewed before you post them to social networks. Different social network pages often require different review processes because the content has different uses and different levels of official involvement from management.

For example, a company that produces and sells smartphones might have several social network accounts:

- A Facebook fans page that requires less supervision and no official review
- A Facebook official page that requires full management approval
- A Twitter official timeline that requires full management approval

When the company launches a new smart phone, the messages that it posts on the social networks require different levels of review.

Using social connected workflows, you can control who creates, edits, approves, and posts messages to social network accounts. For example:

- Content authors can write and post messages for the Facebook fans page without needing a review.
- Content authors can write messages for the official pages, but management should review these messages before the sales team posts the information.

Goal messages

You can use goal messages to automatically post messages to Facebook and Twitter when a website visitor triggers a goal on your website.

This feature requires that you have configured Social Connector on your website.

When a visitor logs in to your website with their social network credentials and performs an action that triggers a goal, the Social Connected module posts a message to the visitor's timeline. The goal messages can be posted to timelines in Facebook and Twitter.

You can modify the message before it is posted to the social network. You can also place a link in the message, and assign a campaign to this link.

Assigning goals and campaigns to goal messages means that you can use Sitecore Analytics to analyze the usage and effectiveness of the goal messages.

You can also personalize a goal message by placing specific information in the message body, for example, the visitor's name or the action that they performed on your website.

Detached messages

A detached message is a message that is not bound to a content item or a goal. You can create, edit, and post detached messages to social networks using the API. Once the message is posted, you can track its campaign statistics in Sitecore Analytics. You can use the API to get statistics from the social network such as the number of Likes, Tweets, and comments.

Direct messages

Direct messages are not assigned to either a Sitecore item or a goal. You can create and post direct messages using the API. In contrast to a detached message, a direct message is not stored in Sitecore and you cannot get its Sitecore Analytics statistics. You post a direct message using API and get back the message identifier from the social network. For a direct message that is posted to Facebook, you can use its identifier to retrieve a list of the comments.

Personalize a goal message

You can personalize a goal message using the API and the UI. The Social Connected module can retrieve information about the visitor or their actions on the website, and you can add this to the goal message.

For example, when a website visitor uploads an image to the website and triggers the uploading image goal, you can transfer the name of the image or visitor's name as goal parameters and then add this information to the message body as tokens.

When you transfer goal parameters using the API, you use the `Sitecore.Social.Client.Api.SocialPageEventClientManager` class and the `TriggerSocialPageEvent` method.

In your custom implementation, use the following code sample to pass the relevant parameter, for example, *username*, to the `social.BuildMessage` pipeline:

```
var parameters = new Dictionary<string, string>
      {
        {
          "username", Sitecore.Context.User.Name
        }
      };
      var socialPageEventClientManager = new SocialPageEventClientManager();
      socialPageEventClientManager.TriggerSocialPageEvent("Login", parameters);
```


Track statistics on a content message

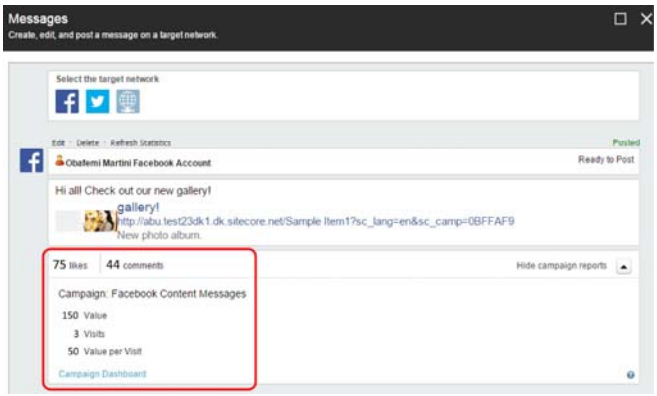
In the Social Connected module, you can track both the campaign and social network statistics on a content message. The campaign statistics are available for messages that have campaigns assigned to them.

A content message uses the default *Twitter Content Messages* and *Facebook Content Messages* campaigns. The reports for these campaigns give you an overview of the effectiveness of all your Twitter or Facebook messages.

To see statistics for a content message:

1. In the Experience Editor, select the content item that have content messages assigned.

2. On the Home tab, click Messages. Under the message, you can see the social network statistics: likes and comments for Facebook and retweets for Twitter messages.
3. Click Show campaign reports under the message that you want to see the statistics on.



4. Click Campaign Dashboard to see the campaign statistics.

Campaign reports and Campaign Dashboard show the statistics for a campaign. If you have this campaign assigned to several messages, you can see the statistics on all the messages. To see the statistics on a particular message, when you create the content message, create a new campaign for it.

Note

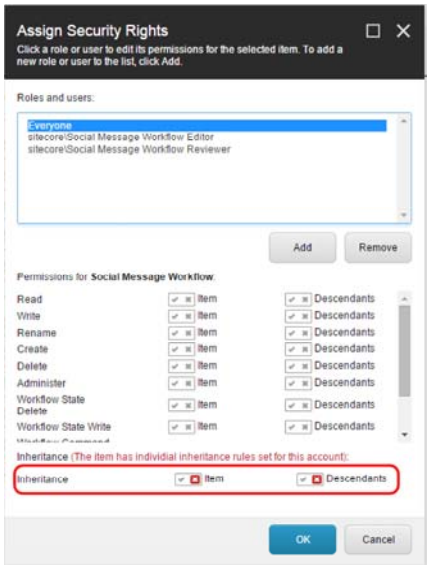
The Social Connected module gathers campaign statistics from xDB aggregate tables. As a result, the campaign statistics are updated as often as the xDB aggregation task runs.

Create a custom workflow

Social Connected module provides two default workflows: the *Social Message* workflow and the *Social Marketer Message* workflow. However, you can also create your own workflow for content messages.

To create and use a custom workflow:

1. Create a Sitecore workflow.
2. In the workflow item, on the Security tab, click Assign. For the *Everyone* role, break the inheritance for the item and its descendants.



3. To separate user access in a workflow by states, commands, and actions, for each type of workflow user, create a Sitecore security role.

   For example, the default *Social Message Workflow Editor* role only has access to the *Draft* state of the *Social Message* workflow.

   - To give users appropriate access to the workflow, specify the following permissions in the Security Editor to the workflow's states, commands, and actions: *Read, Write, Workflow State Delete, Workflow State Write and Workflow Command Execute.*
4. To separate reviewer access in a workflow to messages from a specific social network, create a Sitecore security role for each reviewer.

   For example, the default *Facebook Message Reviewer* role only has access to the Facebook messages of the corresponding Facebook account in Workbox.

5. Create a social network account item.
6. In the Create a Network Account wizard, specify the new workflow and the reviewer roles.
7. Create Sitecore users for authors and reviewers.

- A user that is created for an author must be a member of *sitecore\Author, sitecore\Social Message Author and* assigned the custom role that you have created for this type of workflow user. The custom role grants access to the appropriate states of the workflow. For example, in the default configuration, this is the *Social Message Workflow Editor* role.
- A user that is assigned to a reviewer must be a member of *sitecore\Author and* assigned the roles for the reviewer that you have created for this type of workflow user. The custom reviewer role grants access to the workflow states in which the messages wait for approval and access to the messages of the corresponding social network account. For example, in the default configuration for Facebook accounts, these are the *Social Message Workflow Reviewer* and the *Facebook Message Reviewer* roles.

The default workflows

You can assign a workflow to a specific account when you create an account item. When you create a message and assign it to the account with a workflow, this message is placed in the workflow.

If a message is in a workflow, it must be reviewed before you can post it to a social network.

The Social Connected module provides two default workflows:

- The *Social Message* workflow

  This workflow is for the messages that are posted to social networks.

- The *Social Marketer Message* workflow

  This workflow is for the social marketer messages that are delivered to media marketers who in turn post them on social networks.

You can also create a custom workflow.

Social Message Workflow

The *Social Message* workflow is for content messages that are posted to social networks. It contains three states:

State	Description
Draft	When a content author creates a message, the message is assigned to the *Draft* state. The *Draft* state is also used when the content submitted for review is rejected.
Awaiting Approval	When the message is ready to be posted to a social network, the content author submits it to the *Awaiting Approval* state. The reviewer can either move the message to the *Approved* state or rejects it by moving it back to the *Draft* state.
Approved	When the message is in the *Approved* state, it can be posted to a social network automatically with the content item, or it can be posted manually from the Messages dialog box.

Note

Only the messages in the *Approved* state can be posted to social networks.

The following image shows the schema of the *Social Message* workflow:

Social Marketer Message Workflow

The *Social Marketer Message* workflow is for passing messages to the media marketer who posts the messages to social networks. The *Social Marketer Message* workflow contains four states:

State	Description
Draft	When a content author creates a message, the message is assigned to the *Draft* state.

Awaiting Post Review	When a message is ready to be posted to a social network, the content author submits it to the *Awaiting Post Review* state. The reviewer, usually a media marketer, can • Move the message to the *Post Accepted* state if they accept the message. • Move the message back to the *Draft* state if they do not feel it is ready to be posted, for example, if more information is needed. • Move the message to the *Post Removed* state if they decide that the message does not fit a social network format.
Post Removed	A message in this state is never posted to a social network.
Post Accepted	The reviewer posts the message to a social network.

The following image shows the schema of the *Social Marketer Message*: