Social Connected 1.3 for Sitecore CMS 6.6

A Developer's Guide to Adding a New Social Network

Instructions for configuring a new social network in the Social Connected module

Table of Contents

Chapter 1

Introduction

The Social Connected module connects a Sitecore instance with the major social networks including Facebook, Twitter, LinkedIn, and Google+. You can also configure the Social Connected module to work with any social network that supports the OAuth standard.

This document is designed for administrators and developers. It describes how to configure the module to work with an additional social network.

In this document, we add the Hyves network to the Social Connected module. Hyves is a social networking website in Netherlands with mainly Dutch visitors.

The document contains the following chapters:

- **Chapter 1 — Introduction**
  This introduction to the manual.
- **Chapter 2 — The Basic Configuration**
  This chapter contains the instructions to the required part of the configuration.
- **Chapter 3 — Configuring Social Connector**
  This section describes how to set up the Social Connector feature of the module.
- **Chapter 4 — Configuring Social Messaging**
  This chapter describes how to set up the Social Messaging feature of the module.
- **Chapter 5 — Configure Share Buttons**
  This chapter describes how to set up Share buttons of the module.

1.1 General Configuration Actions

To add an extra social network to the Social Connected module, you must configure the module.

The configuration tasks can be divided into:

- Basic configuration

  This is required and includes:

  - Creating an item, a web application, and the icons for the social network.
  - Creating a Visual Studio project.
  - Creating a configuration file for the social network.
  - Implementing a network provider class.

- Social Connector Configuration

  If the Social Connected module will use the Social Connector feature to log in website visitors, you must:

  - Prepare the connector control.
  - Implement the methods that receive user credentials from the social network.
  - Implement the methods that complete the authorization process.
  - Create a mapping configuration file.

- Social Messaging Configuration

  If the Social Connected module will use the Social Messaging feature to post messages to the social network, you must:

  - Prepare a new account wizard for the social network.
  - Create items for message campaigns.
  - Implement a social network message, provider, and publish provider classes.
  - Implement a method that can post messages to the social network wall.
  - Update a configuration file.
  - Implement the methods that receive social message statistics.

- Share Button Configuration

  If the Social Connected module will use Share buttons with this social network, you must:

  - Create a share button campaign.
  - Create a share button goal.
  - Create a share button sublayout and modifying its markup.
  - Trigger page events.

Note
We recommend that you use a 3rd party component to simplify the requests to the social network API. We use Bee.NET to configure the Hyves network.

Chapter 2

The Basic Configuration

This chapter describes the required configuration actions that you must perform to add a new social network to the Social Connected module.

This chapter contains the following sections:

- Sitecore Configuration
- Visual Studio Project Configuration

2.1 Sitecore Configuration

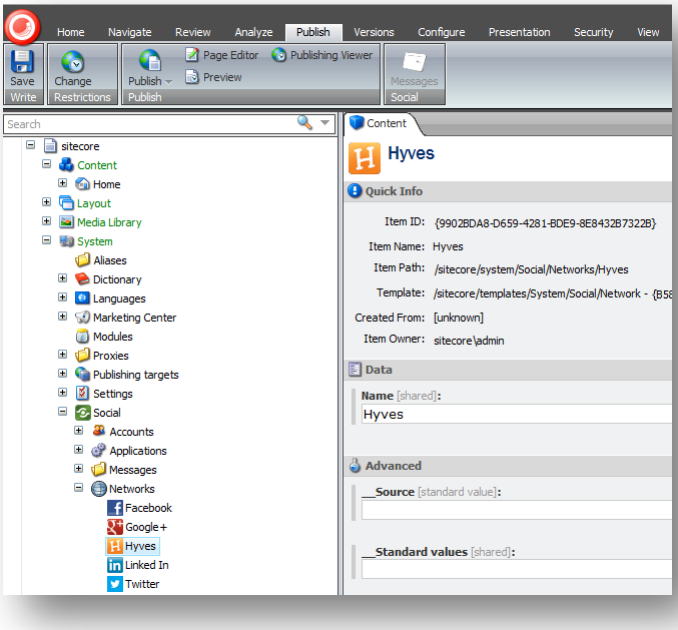
This section describes the required configuration actions in Sitecore to add a new social network to the module.

To perform the required configuration in Sitecore you must:

- Create a social network item.
- Create a social network web application.
- Add social network icons.

2.1.1 Creating a Social Network Item

In the Sitecore **Content Editor**, navigate to the `Sitecore/system/social/networks` folder and create an item based on the `Sitecore/templates/system/social/network` template. Call it, for example, Hyves.

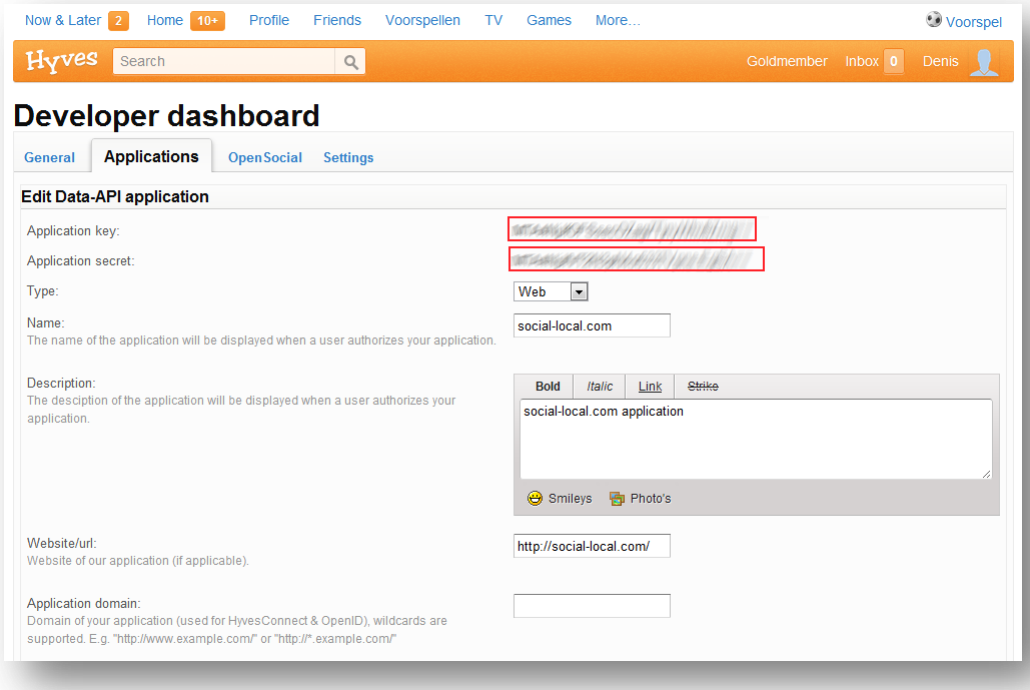


2.1.2 Creating a Social Network Web Application

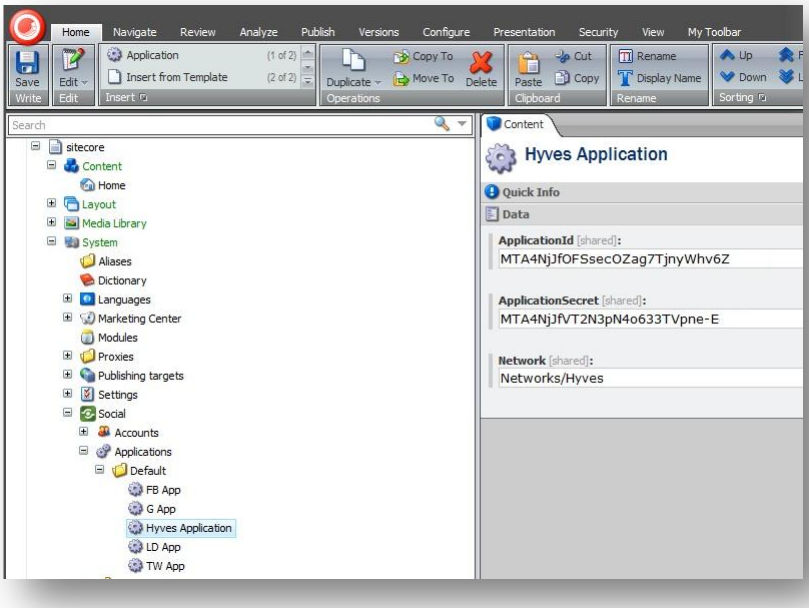
A web application connects the Social Connected module with the social network.

To create a Hyves web application:

1. Go to http://www.hyves.nl/developer/applications/ and create a web application. You must have a developer account to create a web application.



2. Save the values in the **Application key** and the **Application secret** fields in a text file. You need these when creating a Sitecore item for the web application.
3. In the **Content Editor**, navigate to the `Sitecore/system/social/applications/default` folder.
4. In this folder, create an item based on the `Sitecore/templates/system/social/application` template.



5. In the new item, enter the following information:

Field	Value
ApplicationId	The **Application key** of the web application.
ApplicationSecret	The **Application secret** of the web application.
Network	Networks/Social Network Name

2.1.3 Adding Social Network Icons

Create two icons for the new social network. These icons will be rendered in the module UI.

The icons must have the same name. The size of the first icon must be 16 to 16 pixels, and the second must be 32 to 32 pixels. Put the icons in the appropriate folders:

- The `Website\sitecore\shell\themes\standard\custom\16x16` folder.
- The `Website\sitecore\shell\themes\standard\custom\32x32` folder.

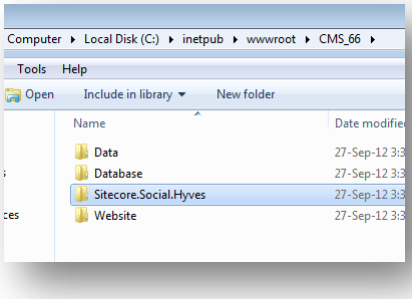
2.2 Visual Studio Project Configuration

To implement business logic of the new social network in the Social Connected module, you must create a Visual Studio project.

2.2.1 Creating a Project in Visual Studio for the Social Network

To create a project in Visual Studio for the social network:

1. In **Visual Studio**, create a new **Class Library** project. Call it, for example, *Sitecore.Social.Hyves*.
2. Add references to the `Sitecore.Kernel` and `Sitecore.Social.Core` libraries.
3. Save this project in the root folder of the CMS instance:



4. Set the following project properties:
   - Configuration — *Release*;
   - Output path — `website_name\website\bin\`

2.2.2 Creating a Configuration File for the Social Network

Create a configuration file for the social network. Call it, for example, `Sitecore.Social.Hyves.config`. Place this file in the `Website\app_config\include` folder.

We recommend that you add a reference to this configuration file in your Visual Studio project.

2.2.3 Implementing a Network Provider Class

To work with the social network you must implement a network provider class.

Create a network provider class that inherits from the `Sitecore.Social.Core.Networks.Providers.NetworkProvider` class. Call it, for example, `Sitecore.Social.Hyves.Providers.HyvesProvider`.

2.2.4 Registering the Social Network in the Configuration File

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 `Sitecore.Social.Hyves.config` file:

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

```
        configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:x="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            <networks>
              <network name="Hyves" ItemId="{9902BDA8-D659-4281-BDE9-8E8432B7322B}"
prefix="hv" icon="hyves" url="http://hyves.nl">
                <providers>
                  <provider type=" Sitecore.Social.Hyves.Providers.HyvesProvider,
Sitecore.Social.Hyves "/>
                </providers>
              </network>
            </networks>
          </sitecore>
        </configuration>
```

The required attributes of the `<network>` node are:

- Name – the network name.
- ItemID – the ID of the network item in Sitecore that you created in the section *Creating a Social Network Item*.
- Prefix – the network prefix.
- Icon – the network icon that you added in the section *Adding Social Network Icons*.
- URL – the URL of the network.

In the `<provider>` node, in the `Type` attribute, enter the class of the social network that you created in the section *Implementing a Network Provider Class*.

Chapter 3

Configuring Social Connector

This section describes how to set up the Social Connector feature of the module.

This chapter contains the following sections:

- Preparing the Connector Control
- Receiving User Credentials
- Completing the Authorization Process

3.1 Preparing the Connector Control

Create a *Login with Social Network* control that contains a button. To find a piece of sample code that you can use, look at the existing controls in the `website\layouts\system\social\connector` folder.

The `OnClick` event must call the `LoginUser` or `AttachUser` methods of the `ConnectUserManager` class:

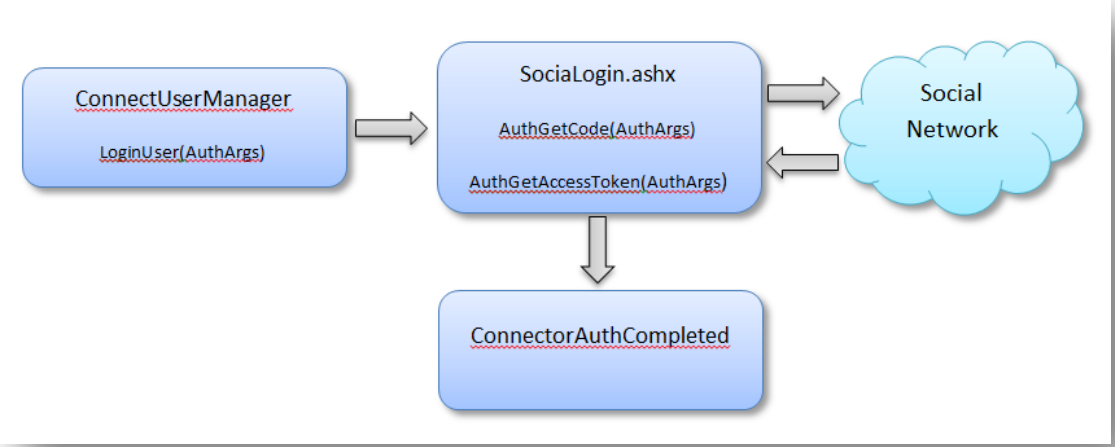
```
            protected void HyvesLoginButtonClick(object sender, ImageClickEventArgs e)
            {
              var connectUserManager = new ConnectUserManager();
              const bool IsAsyncProfileUpdate = true;
              connectUserManager.LoginUser("YourNetwork", IsAsyncProfileUpdate);
        }
```

The parameters of the `LoginUser/AttachUser` method are:

- *YourNetwork* – the network name. This parameter must be the same as the value of the `Name` attribute in the configuration file. For more information about this parameter, see the section *Registering the Social Network in the Configuration File*.
- *isAsyncProfileUpdate* – determines whether or not the profile social data is processed asynchronously.

3.2 Receiving User Credentials

After the `LoginUser/AttachUser` method is called, you must implement the following workflow:



The social network provider class must inherit from the *IAuth* interface, and you must implement two methods:

- `AuthGetCode`
- `AuthGetAccessToken`

3.2.1 Implementing the AuthGetCode Method

The `AuthGetCode` method, compiles the social network authorization URL and transfers it to the social network.

The Hyves authorization URL might look like this:
http://www.hyves.nl/api/authorize/?oauth_token=xxxx&oauth_callback=oauth_callbackURL

The `oauth_callbackURL` parameter of this URL redirects the user from the social network back to the website to complete the authorization process.

The format of the `oauth_callbackURL` parameter is:
http://yourWebSite/layouts/system/Social/Connector/SocialLogin.ashx?type=access&state=authArgsState

The attributes of the `oauth_callbackURL` parameter are:

- *type=access* – this parameter makes `SocialLogin.ashx` call the `AuthGetAccessToken` method;
- *state=authArgsState* – put your `authArgs.StateKey` in this parameter to restore it after the user is returned from the social network.

Sample Code

We implement the `AuthGetCode` method for Hyves using the *Bee.NET* toolkit as follows:

```
            public void AuthGetCode(AuthArgs args)
            {
              var hyvesMethods = new List<HyvesMethod>
              {
                HyvesMethod.All
              };
              var hyvesSession = new HyvesServerSession(args.Application.ApplicationKey,
args.Application.ApplicationSecret, hyvesMethods);
```

```
            var hyvesRequest = new HyvesRequest(hyvesSession);
            string tokenSecret;
            var requestToken = hyvesRequest.CreateRequestToken(out tokenSecret,
HyvesExpirationType.Infinite);

            args.InternalData["requestToken"] = requestToken;
            args.InternalData["requestTokenSecret"] = tokenSecret;

            this.SaveHyvesSession(args, hyvesSession);

            var request = HttpContext.Current.Request;
            var oauthCallback = string.Format("{0}://{1}{2}?type=access&state={3}",
request.Url.Scheme, request.Url.Host, Paths.SocialLoginHandlerPath, args.StateKey);
            var hyvesAuthUrl =
string.Format("http://www.hyves.nl/api/authorize/?oauth_token={0}&oauth_callback={1}",
requestToken, HttpUtility.UrlEncode(oauthCallback));
            HttpContext.Current.Response.Redirect(hyvesAuthUrl);
          }
```

3.2.2 Implementing the AuthGetAccessToken Method

The `AuthGetAccessToken` method:

- Takes the authentication arguments.
- Retrieves the access and secret tokens from the authentication arguments.
- Passes these tokens as parameters to the `CreateAccessToken` method.
- Initializes the `authCompletedArgs` object with the access and secret tokens and the properties of the authentication arguments.
- Invokes the `InvokeAuthCompleted` method.

Sample Code

The following example illustrates how to use the `AuthGetAccessToken` method for Hyves:

```
          public void AuthGetAccessToken(AuthArgs args)
          {
            var requestToken = args.InternalData["requestToken"] as string;
            var requestTokenSecret = args.InternalData["requestTokenSecret"] as string;
            var hyvesSession = this.LoadHyvesSession(args);

            var hyvesRequest = new HyvesRequest(hyvesSession);
            string userId;
            string accessTokenSecret;
            DateTime expireDate;
            var accessToken = hyvesRequest.CreateAccessToken(requestToken,
requestTokenSecret, out accessTokenSecret, out userId, out expireDate);

            var authCompletedArgs = new AuthCompletedArgs
            {
              Application = args.Application,
              AccessToken = accessToken,
              AccessTokenSecret = accessTokenSecret,
              CallbackPage = args.CallbackPage,
              ExternalData = args.ExternalData,
              AttachAccountToLoggedInUser = args.AttachAccountToLoggedInUser,
              IsAsyncProfileUpdate = args.IsAsyncProfileUpdate
            };
            if (!string.IsNullOrEmpty(args.CallbackType))
            {
              this.InvokeAuthCompleted(args.CallbackType, authCompletedArgs);
            }
          }
```

3.3 Completing the Authorization Process

After the module receives the user credentials, it performs the following actions:

- Receiving account basic data.

  Action: implement at least the `GetAccountBasicData` method inheriting from the `IGetAccountInfo` interface.

- User selection.

  The `MatchUser` pipeline runs.

- User logging in.

  Found or created user is logged in.

- Receiving user data from the social network.

  Action: implement at least the `GetAccountInfo` method inheriting from the `IGetAccountInfo` interface.

- Update the user profile.

  The user profile is updated with the received information.

3.3.1 Implementing the GetAccountBasicData Method

To receive the user basic data, you must implement at least the `GetAccountBasicData` method of the `IGetAccountInfo` interface.

The GetAccountBasicData method

```
            public AccountBasicData GetAccountBasicData(Account account)
            {
              var user = this.GetHyvesUser(account);
              return new AccountBasicData
              {
                Account = account,
                Email = null,
                FullName = string.Format("{0} {1}", user.Firstname, user.Lastname),
                Id = user.UserId
              };
            }
```

The GetHyvesUser method

```
            private User GetHyvesUser(Account account)
            {
              var hyvesService = this.GetHyvesService(account);
              return hyvesService.Users.GetLoggedinUser(true);
            }
```

The GetHyvesService method

```
            private HyvesService GetHyvesService(Account account)
            {
              var hyvesService = new HyvesService(this.GetHyvesSession(account));

              return hyvesService;
            }
```

The GetHyvesSession method

```
            private HyvesSession GetHyvesSession(Account account)
            {
              var hyvesSession = new HyvesSession(account.Application.ApplicationKey,
account.Application.ApplicationSecret, new List<HyvesMethod>
```

```
              {
                HyvesMethod.All
              });

              hyvesSession.InitializeToken(account.AccessToken, account.AccessTokenSecret,
DateTime.MinValue);

              return hyvesSession;
            }
```

3.3.2 Implementing the GetAccountInfo Method

To receive user data from the social network, implement the `GetAccountInfo` method.

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. These fields must be declared in the `Sitecore.Social.ProfileMapping.YourSocialNetwork.config` file.

  For more information about how to create this configuration file, see the section *Creating a Mapping Configuration File*.

Sample Code

Some of the fields that the module receives from the social network have a common request method. We group the fields according to method, call this method, and then parse the data for each field in the group:

```
            public IEnumerable<Field> GetAccountInfo(Account account, IEnumerable<FieldInfo>
acceptedFields)
            {
              var hyvesSession = this.GetHyvesSession(account);

        // receives the method dictionatyt where key: method name (described in config),
value: method enum value
              Dictionary<string, HyvesMethod> methodDescriptionDictionary =
this.GetMethodDescriptionDictionary();

              var fieldsGroupedByAccess = acceptedFields.Where(field => field["method"] !=
null)
                    .GroupBy(field => field["method"])
                    .Select(fg => new { Method = fg.Key, Fields = fg.ToList() });

              foreach (var groupFields in fieldsGroupedByAccess)
              {
                var request = new HyvesRequest(hyvesSession);
                var hyvesResponse =
request.InvokeMethod(methodDescriptionDictionary[groupFields.Method], false);

                // You'll find documentation here:
                // http://james.newtonking.com/projects/json/help/
                var jObject = JObject.Parse(hyvesResponse.RawResponse);

                foreach (var field in groupFields.Fields)
                {
                  var token = jObject.SelectToken(field["selectToken"]);
                  var value = (token != null) ? token.ToString() : null;
                  if (!string.IsNullOrEmpty(value))
                  {
                    yield return new Field { Name = field.SitecoreKey, Value = value};
                  }
                }
              }
            }
```

3.3.3 Creating a Mapping Configuration File

To map the information in the social network user profile to a Sitecore database, create a mapping configuration file.

To create a mapping configuration file:

1. In the `website\app_config\include` folder, create a text file. Call it, for example, `Sitecore.Social.ProfileMapping.Hyves.config`.
2. In the `Sitecore.Social.ProfileMapping.Hyves.config` file, create a `sitecore/socialProfileKeyMappings/network` section with the `Name` attribute.
3. In the `network` section, add some `field` sections, that contain the following required attributes:
    - Enabled — it receives information for this field from the network. Possible values: true or false.
    - OriginalKey — the field name in the social network database.
    - SitecoreKey — the field name in the Sitecore user profile.
    - Text — the display name of the field that is rendered in the module UI.

The `field` section contains some 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 this extra data.

In the following sample code, we use two optional field values:

- Method — the API method name to call;
- selectToken — the path to parse in the JSON response to receive the field values.

`Sitecore.Social.ProfileMapping.Hyves.config`:

```
       <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:x="http://www.sitecore.net/xmlconfig/">
         <sitecore>
           <socialProfileKeyMappings>
             <network name="Hyves">
               <field enabled="true" originalKey="userid" selectToken="user[0].userid"
sitecoreKey="hv userid" method="users.getLoggedin" text="User Id"/>
               <field enabled="true" originalKey="displayname"
selectToken="user[0].displayname" sitecoreKey="hv displayname" method="users.getLoggedin"
text="Display Name"/>
               <field enabled="true" originalKey="firstname" selectToken="user[0].firstname"
sitecoreKey="hv_firstname" method="users.getLoggedin" text="First Name"/>
               <field enabled="true" originalKey="lastname" selectToken="user[0].lastname"
sitecoreKey="hv lastname" method="users.getLoggedin" text="Last Name"/>
               <field enabled="true" originalKey="gender" selectToken="user[0].gender"
sitecoreKey="hv_gender" method="users.getLoggedin" text="Gender"/>
               <field enabled="true" originalKey="birthday" selectToken="user[0].birthday"
sitecoreKey="hv birthday" method="users.getLoggedin" text="Birthday"/>
             </network>
           </socialProfileKeyMappings>
         </sitecore>
       </configuration>
```

Chapter 4

Configuring Social Messaging

This chapter describes how to set up the Social Messaging feature of the module.

This chapter contains the following sections:

- Sitecore Configuration
- Visual Studio Project Configuration

4.1 Sitecore Configuration

This section describes how to configure social messaging in Sitecore.

To configure social messaging, you must:

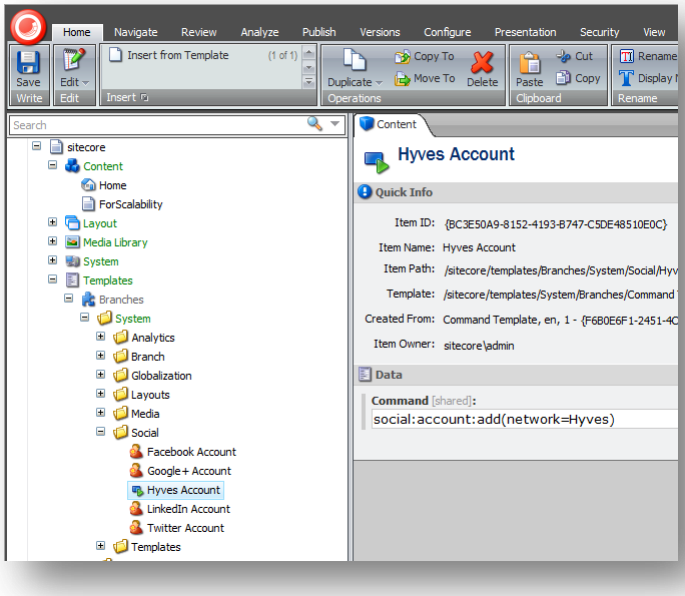
- Create a New Account wizard.
- Create a social message template.
- Create the campaign items for the social network messages.

4.1.1 Creating a New Account Wizard

You must create a New Account wizard. Administrators can create items for the social network accounts using this wizard.

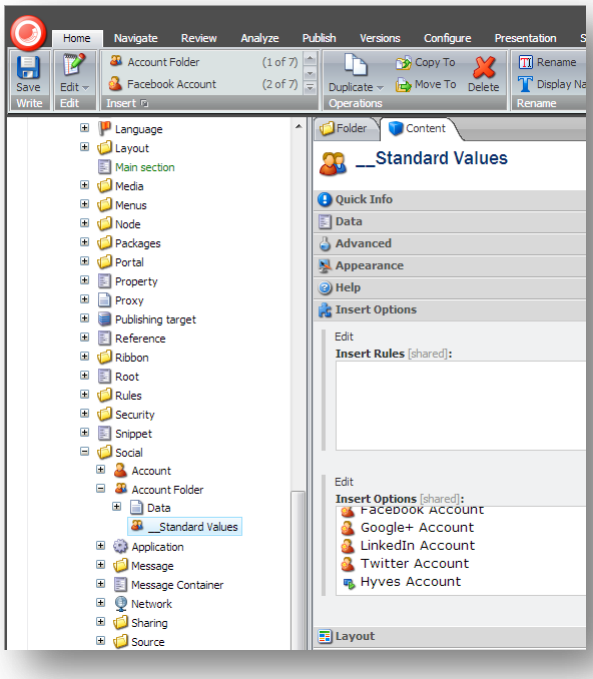
To create a wizard for the social network account:

1. In the **Content Editor**, in the `Sitecore/templates/branches/system/social` folder, create a new item, based on the `Sitecore/templates/system/branches/command template` template. Call it, for example, *Hyves Account*.



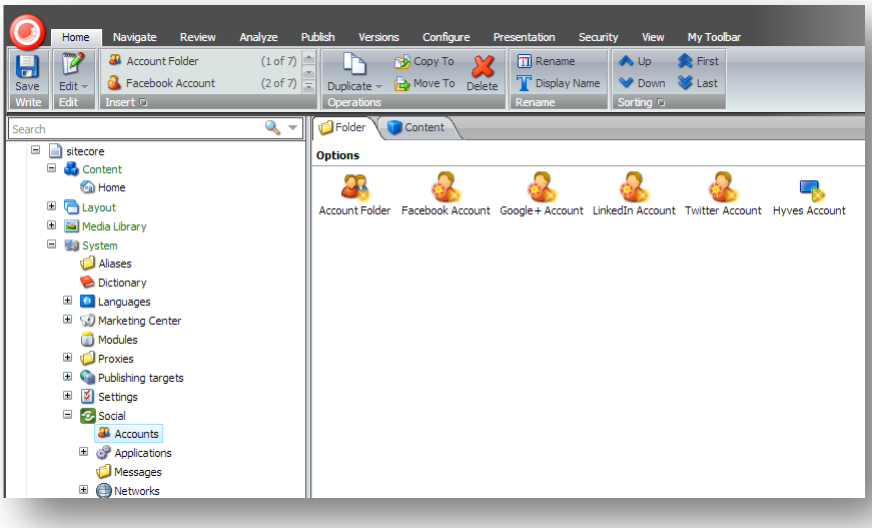
2. In the *Hyves Account* item, in the **Command** field, enter *social:account:add(network=Hyves)*.

3. Navigate to the `Sitecore/templates/system/social/account folder` item.



4. In the *__Standard Values* subitem, in the **Insert Options** field, add the `templates/branches/system/social/hyves account` item.

As a result, the **Hyves Account** wizard is added to the `Sitecore/system/social/accounts` folder:

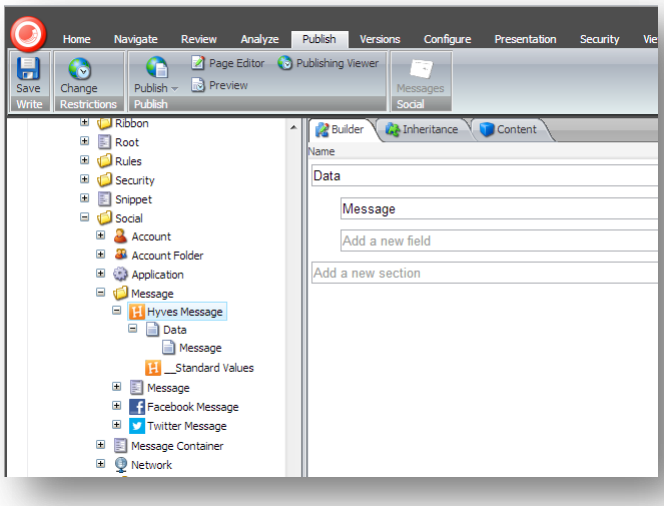


4.1.2 Creating a Social Message Template

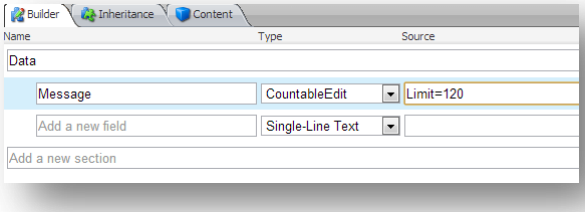
Social messages are stored in the message items that are based on the social message template.

To create a social message template:

1. In the **Content Editor**, in the `Sitecore/templates/system/social/message` folder, create a new template based on the `/sitecore/templates/system/social/message/message` template.
2. In the **Data** section, add the data fields that social network message should contain, for example a **Message** field



3. To limit the length of a message, in the **Message** field:
   - Select the **CountableEdit** type.
   - In the **Source** field, set the **Limit** attribute.



4. In the social network message item, create the **__Standard Values** item.

4.1.3 Creating Campaign Items

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

To create the campaign items for the social network messages:

1. In the **Content Editor**, in the `Sitecore/system/marketing center/campaigns/social` folder, create a new item based on the `sitecore/templates/system/analytics/campaign category` template. Call the new item, for example, *Hyves*.
2. In the *Hyves* item, create a new item based on the `sitecore/templates/system/analytics/campaign category` template. Call the new item, for example, *Hyves Message Campaigns*.

   This item will store the custom campaigns that will use the social messages.

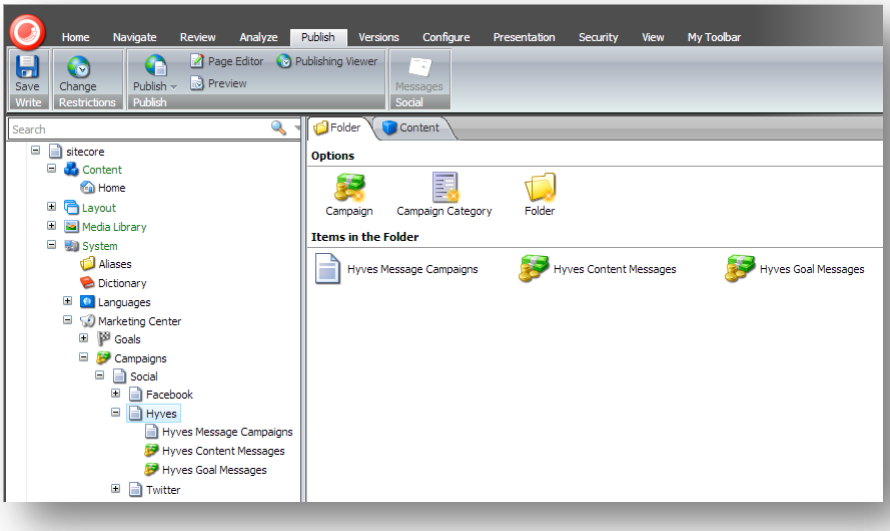
3. In the *Hyves* item, create a new item based on the `sitecore/templates/system/analytics/campaign` template. Call the new item, for example, *Hyves Content Messages*.

   This is the default campaign for content messages.

4. In the *Hyves* item, create a new item based on the `sitecore/templates/system/analytics/campaign` template. Call the new item, for example, *Hyves Goal Messages*.

   This is the default campaign for goal messages.

The item structure should look like this:



4.2 Visual Studio Project Configuration

This section describes how to configure social network messaging in a Visual Studio project.

4.2.1 Implementing the Social Network Message Class

Implement the `Social Network Message Class` that is inherited from the `Sitecore.Social.Core.Publishing.Items.SocialMessageBase` class. The `Social Network Message Class` creates the Sitecore items for a social network message.

Sample Code

The following code illustrates how to implement the `HyvesMessage` class for Hyves:

```
/// <summary>
/// The class represents the wrapper on the Hyves message item.
/// </summary>
public class HyvesMessage : SocialMessageBase
{
  private const string MessageFieldName = "Message";

  private const string LinkFieldName = "Link";

  private string message;

  private string link;

  public HyvesMessage()
  {

  }

  public HyvesMessage(Item item) : base(item)
  {

  }

  public string Message
  {
    get
    {
      return this.messageItem.Fields[MessageFieldName].Value;
    }

    set
    {
      this.message = value;
    }
  }

  public string Link
  {
    get
    {
      return this.messageItem.Fields[LinkFieldName].Value;
    }

    set
    {
      this.link = value;
    }
  }

  /// <summary>
  /// Saves the data to the item.
  /// </summary>
  public override void SaveData()
  {
    base.SaveData();
```

```
            this.messageItem.Editing.BeginEdit();
            this.messageItem.Fields[LinkFieldName].Value = this.link;
            this.messageItem.Fields[MessageFieldName].Value = this.message;
            this.messageItem.Editing.EndEdit();
          }
        }
```

4.2.2 Implementing the Social Network Renderer Class

Implement the `Social Network Renderer` class that inherits from the `Sitecore.Social.Core.Publishing.Renderers.IMessageRenderer` interface. The `Social Network Renderer` renders social network messages.

Sample Code

The following code illustrates how to implement the `HyvesMessageRenderer` class for Hyves:

```
        public class HyvesMessageRenderer : IMessageRenderer
        {
          /// <summary>
          /// Renders the message.
          /// </summary>
          /// <param name="socialMessageBase">The social message base.</param>
          /// <param name="sourceName">Name of the source.</param>
          /// <param name="messageTextRenderStrategy">The message text render
strategy.</param>
          /// <returns>
          /// The rendered html.
          /// </returns>
          public string RenderMessage(SocialMessageBase socialMessageBase, string
sourceName, IMessageTextRenderStrategy messageTextRenderStrategy)
          {
            var hyvesMessage = new HyvesMessage(socialMessageBase.MessageItem);

            var stringBuilder = new StringBuilder();

            var anchorTag = string.Format(CultureInfo.CurrentCulture, @"<a href=""{0}""
target=""_blank"">{0}</a>",
Social.Core.Publishing.Managers.LinkManager.GenerateLink(hyvesMessage.Link,
hyvesMessage.CampaignId));

            var messageText =
messageTextRenderStrategy.Render(socialMessageBase.MessageItem.ID, sourceName,
HttpUtility.HtmlEncode(hyvesMessage.Message));

            stringBuilder.Append(messageText);

            if (messageText.Contains("$link"))
            {
              stringBuilder = stringBuilder.Replace("$link", anchorTag);
            }
            else
            {
              stringBuilder.Append("</br>");
              stringBuilder.Append(anchorTag);
            }

            return stringBuilder.ToString();
          }

          /// <summary>
          /// Gets the message content CSS class.
          /// </summary>
          /// <returns>
          /// The css class name.
          /// </returns>
          public string GetMessageContentCssClass()
          {
            return string.Empty;
          }
        }
```

4.2.3 Implementing the Social Network Publish Provider Class

Implement the `Social Network Publish Provider` class that inherits from the `Sitecore.Social.Core.Publishing.Providers.PublishProviderBase` class.

The `Social Network Publish Provider` class should:

- Override the `PublishAll` method.
- In the `PublishAll` method, call the `Publish` method of the base `PublishProviderBase` class and implement message posting for a specific account in an anonymous function.
- In the anonymous function, prepare the account data that you want to post and call the `PublishOnTheWall` method of the appropriate network provider.

Sample Code

The following code illustrates how to implement the `HyvesPublishProvider` class for Hyves:

```
          /// <summary>
          /// The class prepares message for posting and posts in on the Hyves WWW (Who What
Where)
          /// </summary>
          public class HyvesPublishProvider : PublishProviderBase
          {
            private const string AccessTokenFieldName = "AccessToken";
            private const string AccessTokenSecretFieldName = "AccessTokenSecret";
            private const string MessageFieldName = "Message";
            private const string LinkFieldName = "Link";

            public HyvesPublishProvider(Message message) : base(message)
            {
            }

            /// <summary>
            /// Posts the message on the Hyves WWW.
            /// </summary>
            public override void PublishAll()
            {
              this.Publish(account =>
              {
                var applicationKey = (string)null;
                var applicationSecret = (string)null;
                var accessToken = account[AccessTokenFieldName];
                var accessTokenSecret = account[AccessTokenSecretFieldName];

                if (account.Application != null)
                {
                  applicationKey = account.Application.ApplicationKey;
                  applicationSecret = account.Application.ApplicationSecret;
                }

                var networkApplication = new Application
                {
                  ApplicationKey = applicationKey,
                  ApplicationSecret = applicationSecret
                };

                var networkAccount = new Account
                {
                  AccessToken = accessToken,
                  AccessTokenSecret = accessTokenSecret,
                  Application = networkApplication
                };

                var hyvesProvider = new HyvesProvider(networkApplication);

                var hyvesMessage = this.BuildHyvesMessage(this.Message);

                var messageId = hyvesProvider.PublishOnTheWall(networkAccount, new
Social.Core.Networks.Messages.Message
                {
```

```
                Text = hyvesMessage
              });

              return new PostResult { MessageId = messageId, Responce = new
Dictionary<string, string>() };
            });
          }

          /// <summary>
          /// Builds the hyves message.
          /// </summary>
          /// <param name="message">The message.</param>
          /// <returns></returns>
          protected virtual string BuildHyvesMessage(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();
          }
        }
```

4.2.4 Implementing the PublishOnTheWall Method

To post messages on the social network wall, you must inherit the network provider class from the `IPublishOnTheWall` interface and implement the `PublishOnTheWall` method. This method must return a new message ID.

Sample Code

The following code illustrates how to implement the `PublishOnTheWall` method for Hyves:

```
          public string PublishOnTheWall(Account account, Message message)
          {
            var accountId = this.GetAccountId(account);

            var hyvesService = this.GetHyvesService(account);
            hyvesService.Session.InitializeUserId(accountId);
            var www = hyvesService.Wwws.CreateWww(message.Text, HyvesVisibility.SuperPublic,
null, null, null, null, null);
            return www.WwwId;
          }
```

4.2.5 Updating the Social Network Configuration File

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

Sitecore.Social.Hyves.Config

```
       <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:x="http://www.sitecore.net/xmlconfig/">
         <sitecore>
           <networks>
             <network name="Hyves" ItemId="{21F729AE-6731-43B5-A131-749216780C75}"
prefix="hv" icon="hyves" url="http://hyves.nl">
               <items>
                 <message type="Sitecore.Social.Hyves.Publishing.Items.HyvesMessage,
Sitecore.Social.Hyves"
                          MessageTemplateId="{319D7C4F-AB38-4CBA-80A4-10BC852C7AB8}"

Renderer="Sitecore.Social.Hyves.Publishing.Renderers.HyvesMessageRenderer,
Sitecore.Social.Hyves"/>
               </items>
               <publishing>
                 <campaigns rootCampaignItemId="{1BD24FDE-C3EC-479A-85CB-D3F919D5F925}">
                   <campaign source="Goal" itemId="{5C3C2357-BB7A-446F-A169-E1B9B381AD5F}"/>
                   <campaign source="Publish" itemId="{B9399D02-14BF-42EE-983B-
EC8D28216D03}"/>
                 </campaigns>
                 <publisher
type="Sitecore.Social.Hyves.Publishing.Providers.HyvesPublishProvider,
Sitecore.Social.Hyves"/>
               </publishing>
               <providers>
                 <provider type="Sitecore.Social.Hyves.Providers.HyvesProvider,
Sitecore.Social.Hyves"/>
               </providers>
             </network>
           </networks>
         </sitecore>
       </configuration>
```

The configuration attributes are:

- `rootCampaignItemId` – the ID of the social network message campaign item.
- <campaign source="Publish"> – the ID of the social network content message item.
- <campaign source="Goal"> – the ID of the social network goal message item.

4.2.6 Receiving Message Statistics

To receive message statistics from the social network, you must inherit the social network provider from the `Sitecore.Social.Core.Networks.Providers.Interfaces.IMessageStatistics` interface and implement the following methods:

- `List<string> StatisticNames { get; }` – this method returns the collection of the statistics names.
- `Dictionary<string, double> GetMessageStatistics(Account account, string messageId)` – this method returns the message statistics.
- `string GetStatisticsCounterDisplayName(string statisticsCounterName)` – this method returns the display name of the statistics counter.

Sample Code

```
           public List<string> StatisticNames
           {
             get
             {
               return new List<string> { "Respects", "Comments" };
             }
           }
           public Dictionary<string, double> GetMessageStatistics(Account account, string
messageId)
```

```
            {
              var accountId = this.GetAccountId(account);

              var hyvesService = this.GetHyvesService(account);
              hyvesService.Session.InitializeUserId(accountId);

              var respects = hyvesService.Wwws.GetRespects(messageId);
              var comments = hyvesService.Wwws.GetComments(messageId);

              var result = new Dictionary<string, double>();

              if (respects != null)
              {
                result.Add(RespectsCounterKey, respects.Count);
              }

              if (comments != null)
              {
                result.Add(CommentsCounterKey, comments.Count);
              }

              return result;
            }

            public string GetStatisticsCounterDisplayName(string statisticsCounterName)
            {
              if (string.Compare(RespectsCounterKey, statisticsCounterName,
StringComparison.CurrentCultureIgnoreCase) == 0)
              {
                return Translate.Text(Common.Texts.Respects);
              }

              if (string.Compare(CommentsCounterKey, statisticsCounterName,
StringComparison.CurrentCultureIgnoreCase) == 0)
              {
                return Translate.Text(Common.Texts.Comments);
              }

              return statisticsCounterName;
            }
```

Chapter 5

Configure Share Buttons

This chapter describes how to set up Share buttons of the module.

This chapter contains the following sections:

- Creating a Share Button Campaign
- Creating a Share Button Goal
- Creating a Share Button Parameters Template
- Creating a Share Button Sublayout
- Creating a Share Button Sublayout Markup
- Tracking Share Button Events

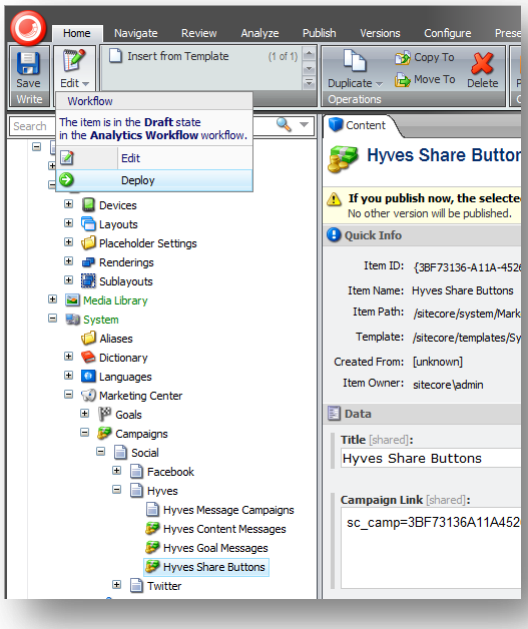
5.1 Creating a Share Button Campaign

When the website visitor clicks a Share button, an appropriate message with the link to the website is displayed in the social network. After any visitor clicks that link, the appropriate campaign is triggered.

To create a campaign for a share button:

1. In the **Content Editor**, navigate to the `/sitecore/system/marketing center/campaigns/social` folder.
2. In this folder, create the *Hyves* folder.
3. In the *Hyves* folder, create a new item based on the `sitecore/templates/system/analytics/campaign` template.

   Call the new item, for example, *Hyves Share Buttons*.
4. DMS 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**.

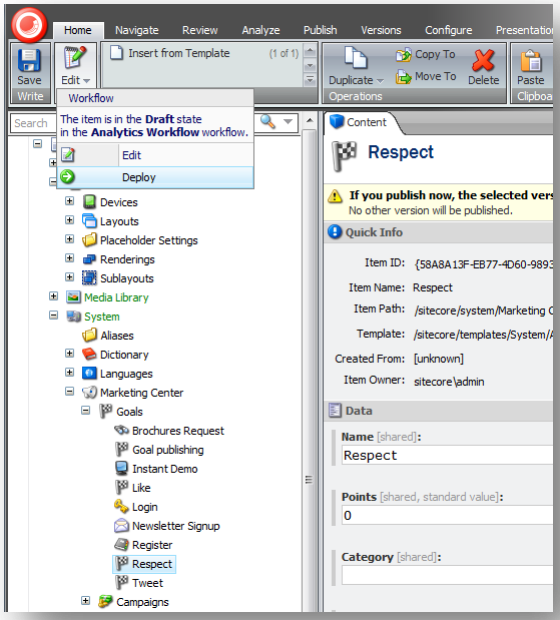


5.2 Creating a Share Button Goal

When the website visitor clicks a Share button, an appropriate message with the link to the website is displayed in the social network. After any visitor clicks that link, the appropriate goal is triggered.

To create a goal for a share button:

1. In the **Content Editor**, navigate to the `/sitecore/system/marketing center/goals` folder.
2. Create a new goal. Call it, for example, *Respect*. In the Hyves network, a share button is called the *Respect* button.
3. DMS 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**.

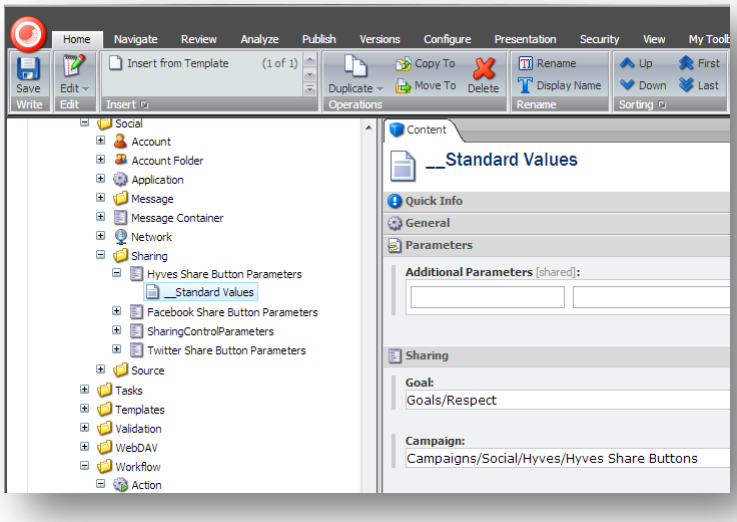


5.3 Creating a Share Button Parameters Template

A social share button is a Sitecore sublayout. You can create a parameters template to pass the parameters to the sublayout.

To create a parameters template for a share button:

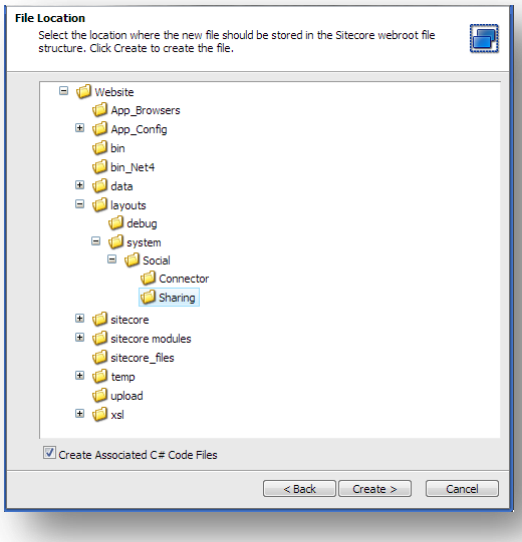
1. In the **Content Editor**, navigate to the `/sitecore/templates/system/social/sharing` folder.
2. Create a new template with the following parameters:
   - Name – *Hyves Share Button Parameters*;
   - Base template – `templates/system/social/sharing/SharingControlParameters`;
3. Create a *__Standard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



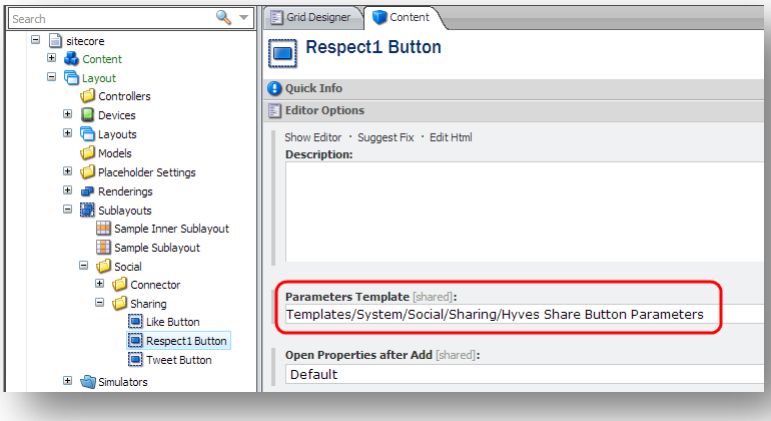
5.4 Creating a Share Button Sublayout

To create a share button sublayout:

1. In the **Content Editor**, navigate to the `/sitecore/layout/sublayouts/social/sharing` folder.
2. Create a new sublayout with the following parameters:
   - Name – *Respect Button*;
   - Item Location – `sublayouts/social/sharing`;
   - File Location – `website/layouts/system/social/sharing`;
   - Create the associated C# code files.



3. In the *Respect Button* item, in the **Parameters Template** field, select the *Share Button Parameters* template that you created in the previous section.

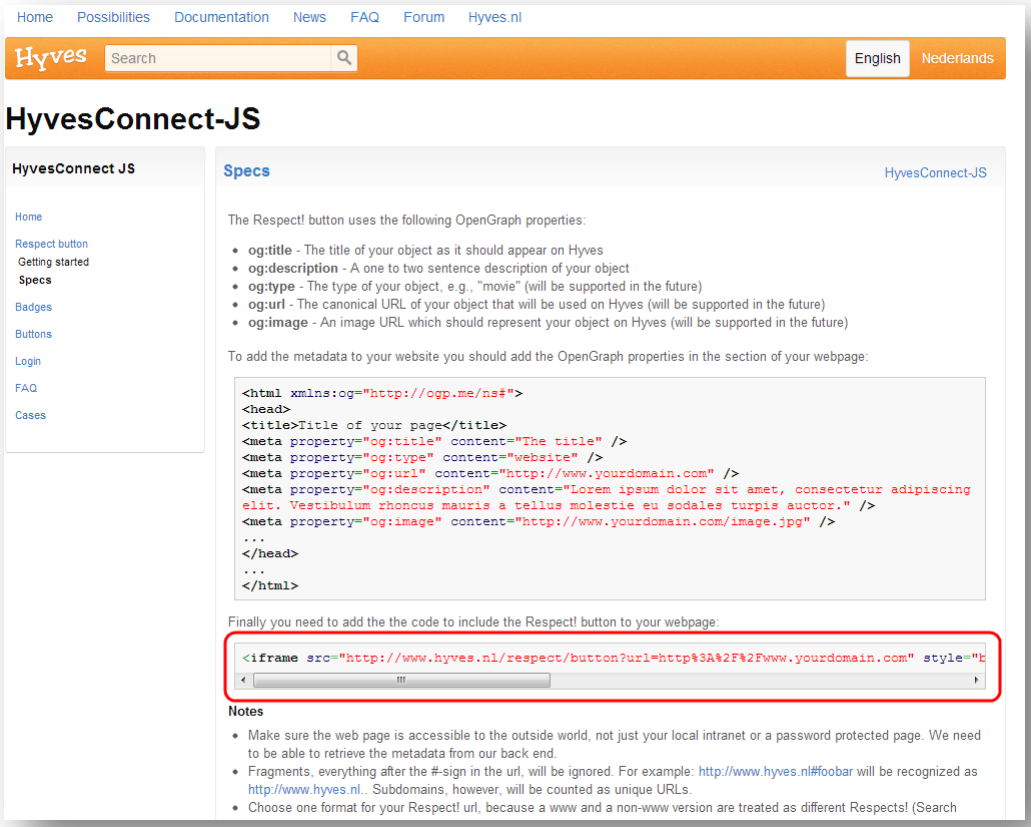


5.5 Creating a Share Button Sublayout Markup

A share button sublayout markup is empty by default. You must write its markup.

To create a sublayout markup of a share button:

1. Go to http://www.hyves-developers.nl/documentation/hyvesconnect-js/specs-respect
2. Copy the code:



3. Open the `website_name\website\layouts\system\social\sharing\Respect Button.ascx` file.
4. In the `Respect Button.ascx` file:
   - Paste the code that you copied from the Hyves website.
   - Replace the value of the `url` parameter with:

```
<%=HttpUtility.UrlEncode(GetCampaignQS()) %>
```

   The `Respect Buttons.ascx` file should now look like this:

```
<%@ Control Language="c#" AutoEventWireup="true"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5"
Inherits="Layouts.Respect_button.Respect_buttonSublayout"
CodeFile="~/layouts/system/Social/Sharing/Respect Button.ascx.cs" %>
    <iframe
src="http://www.hyves.nl/respect/button?url=<%=HttpUtility.UrlEncode(GetCampaignQS()) %>"
style="border: medium none; overflow:hidden; width:150px; height:21px;" scrolling="no"
frameborder="0" allowTransparency="true" ></iframe>
```

5. Open the `website_name\website\layouts\system\social\sharing\Respect Button.ascx.cs` file.

6. In this file, edit the code so that it looks like this:

```
using System;
using Sitecore.Social.Client.Sharing.Controls;

namespace Layouts.Respect_button

 /// <summary>
 /// Summary description for Respect1_buttonSublayout
 /// </summary>
  public partial class Respect_buttonSublayout : ShareButtonBase
 {
    private void Page Load(object sender, EventArgs e) {
      // Put user code to initialize the page here
    }
  }
}
```

5.6 Tracking Share Button Events

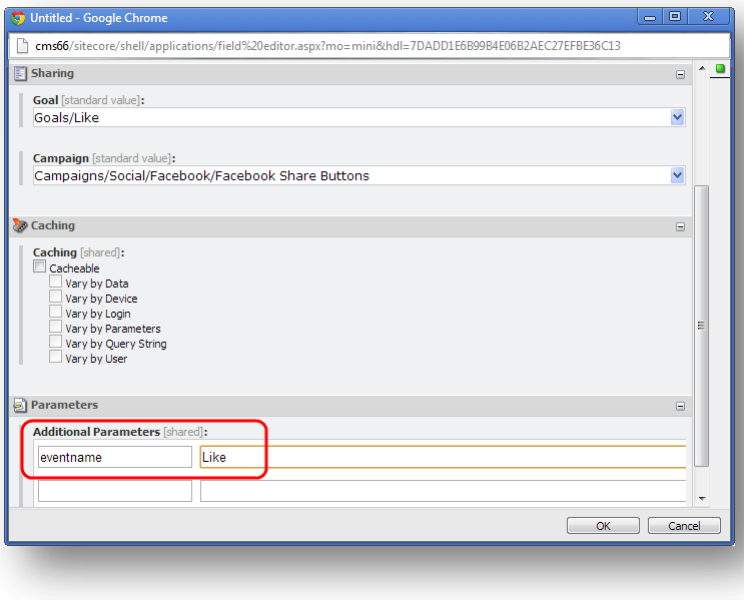
The Hyves network does not support the tracking of share button events. We will therefore describe how to do it on the Facebook Like button.

The following code in the `LikeButton.ascx.cs` file is responsible for triggering page events:

```
        public partial class LikeButton : ShareButtonBase
          {
            protected void Page_Load(object sender, EventArgs e)
            {
              const string RequestScript = "var xmlhttp;if (window.XMLHttpRequest){
xmlhttp=new XMLHttpRequest();}else{xmlhttp=new
ActiveXObject(\"Microsoft.XMLHTTP\");}xmlhttp.open(\"GET\",url,true);xmlhttp.send();";

              var script = string.Format(
                CultureInfo.CurrentCulture,
                @"<script>FB.Event.subscribe('edge.create', function (response) {{var url =
'{1}?itemid={0}&eventname={2}&goalname={4}';{3}}});</script>",
                Sitecore.Context.Item.ID,
                SharingHelper.ServicePagePath,
                HttpUtility.UrlEncode(this.EventName),
                RequestScript,

HttpUtility.UrlEncode(Sitecore.Configuration.Settings.GetSetting("Social.DefaultLikeGoal")));

              this.Page.ClientScript.RegisterStartupScript(this.GetType(),
"facebooksubscribe", script);
            }
          }
```

There are three parameters that you must specify in this file:

- itemid – the ID of the item where is the Like button is placed.
- eventname – the name of the goal that is triggered when Like is counted by Facebook. This parameter is not required. You can use it when a special goal must be triggered by a share button.



- goalname – the name of the goal that is triggered by share buttons. This parameter is not required. You can specify both the `eventname` and the `goalname` parameters or only one of them.