Integrating Facebook with your KnowledgeKube Application

This short guide will teach you how to create a KnowledgeKube application that allows users to log in and access their own Facebook account. The implementation described in this guide includes a combination of the Facebook **Data Provider**, a **Custom Provider** for retrieving access tokens, a **Redirect** to the Facebook site, and the *UpdateConnectionString* function to override the provider connection with retrieved details about the current user.

Step 1

First you'll need to create an application inside Facebook. More information about this process can be found here:

https://developers.facebook.com/

Once you've created your "app" you will be given an **Application ID** and **App Secret**. You will need to retain these, as you will be using them later to make requests to Facebook from your KnowledgeKube application. Open the **Basic Settings** section for your Facebook app and enter your website domain (e.g. *yoursandboxname.onknowledgekubesandbox.co.uk*) in its **App Domains** and **Site URL** field. You can also set a category, add an icon and change various other settings here. It's probably worth adding a test user, which you can do via the **Roles** menu.

During development, your new app will be in **Test Mode**. After you have fully configured your app and the associated KnowledgeKube functionality, you can make the app public. Depending on the permissions your app requires, Facebook may need to review your application before it is available for public use. For this example, we will be running the app in test mode only.

Step 2

Read through the document on the Facebook Login process and familiarise yourself with how this works. The example in this guide implements part of the necessary flow, but you could extend your model to incorporate further features.

https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow

Step 3

You'll now need to create the model content necessary to integrate with your Facebook app. To access a user's data, you'll need to create a connection using the **Facebook Data Provider**. In the connection's **Provider Settings**, enter the *OAuth Client Id* and *OAuth Client Secret* values you obtained when configuring your app. For the *OAuth Access Token* property, type a value of 0. This generic value is a placeholder that we'll be replacing with a user's genuine access token during run-time.

Using the new connection, create a data source called *Posts_DS* that connects to the target database's *Posts* table. When a connection is established using a real access token, the connection will be established with the specified user's *Posts* table, allowing your site to draw down data or write a new post, as your requirements dictate.

Step 4

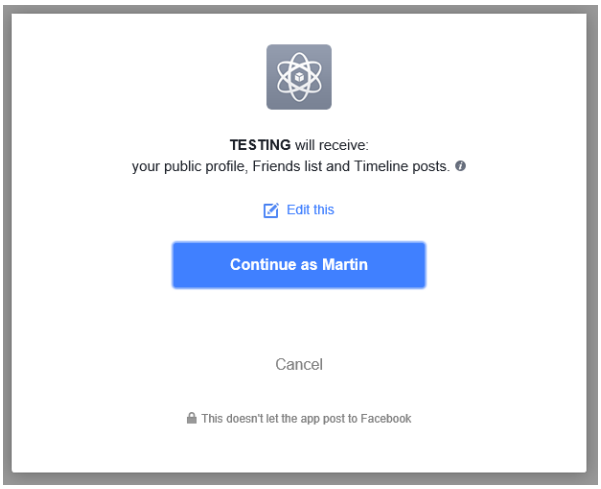
Create three **Constants**, setting each one's value to what is described below:

- *ClientId* – the *Application ID* of your app.
- *ClientSecret* – your app's *App Secret*.
- *RedirectUri* – the URL of the page you want the user to return to after logging into Facebook.

Next, create a button somewhere in your model. When clicked, this button will redirect users to Facebook. The expression you'll need in order to implement this will look something like the one below:

```
RedirectToURL(FormatString("https://www.facebook.com/v2.8/dialog/oauth?client_id={0}&redirect_uri={1}&scope=user_posts","ClientId,RedirectUri"));
```

This expression supplies Facebook with the value of *ClientId*, allowing it to identify and load your app. When the app loads, the user will be asked to enter their user name and password. After logging in, users will be prompted to allow your app to access their details. Below is a screenshot illustrating what a typical user will see after they log in:



Whether or not they agree to this, the app will then redirect the user back to the URL contained in *RedirectUri*.

The expression also includes a **Scope** parameter, which lets you specify the permissions required by your app. Our expression requests access to the authenticated user's posts by specifying *user_posts*, but you could expand this by adding more values to the parameter and separating them with commas. To see a list of the permissions you can request, refer to the following link:

https://developers.facebook.com/docs/facebook-login/permissions

Step 5

If the user agrees to share their details with your app, Facebook will place a special code parameter in the URL used to redirect them back to your site. Your site will need to send this code to Facebook for an **Access Token**, which can then be used to authorise any requests to Facebook for the user's data – more on that in step 7.

To retrieve the code from the return URL, you will need to write a **Form Load Expression** that calls the *GetQueryStringText* function, and ensure this expression is called as soon as the user returns to your site. Using the following expression we can take the value of the *code* parameter and assign it to a run-time variable, also named *Code*.

```
Code:=GetQueryStringText("code");
```

Step 6

The end user may decide not to share their account information with your app, in which case the return URL will contain an error code. You can handle this scenario by writing a conditional statement at the top of the new Form Load Expression to test whether the error code (in this case, 200) was returned and, if so, to redirect the user to a different section of your site. In this example, we'll redirect them to the *Cancelled* question group.

```
if:(GetQueryStringText("error_code") = 200)
{
        ShowForm("Cancelled");
}
```

Step 7

Assuming the user has granted permission to your app, you'll need to create a method for passing the value of *Code* to Facebook in exchange for a valid access token. To do this, you'll need to create a **Custom Provider** that includes a driver item of type **Stored Procedure**. For our example, we'll call this item *GetAccessToken*. If you aren't sure how to set this up, refer to the separate document titled "**Creating a Custom Provider to Fetch an Access Token from Facebook"**.

After you create your provider, add a **Connection** to it. Since all of the necessary details will be input directly into the stored procedure, you don't need to make any changes to the connection's default properties.

Next, create a KnowledgeKube **Stored Procedure** named *GetAccessToken* that uses your new connection. Make sure you assign aliases to each of the input parameters, such that each alias is named after the parameter itself.

Step 8

To make sure *Code* contains a suitable value, you'll need to add a conditional statement to the end of your Form Load Expression. If the value of *Code* is valid (in this case, it is more than one character in length), the expression should use the *RunStoredProcedure* function to activate your new stored procedure and pass the resulting string to the *AccessToken* run-time variable.

Next, use the *FormatString* function to construct a property pair string starting with "OAuth Access Token=" and ending with the value of *AccessToken*. Finally, use the *UpdateConnectionString* function to update the *Posts_DS* connection so it uses the value of *AccessToken* in place of the connection's default, generic token.

```
if:(strlen(Code) > 1)
{
        AccessToken:=RunStoredProcedure("GetAccessToken","ClientId,RedirectUri,ClientSecret,Code",2);

        Properties:=FormatString("OAuth Access Token={0}","AccessToken");

        UpdateConnectionString("Posts_DS",Properties);
}
```

After this process occurs, the *Posts_DS* data source will be tied to the specific user's account for the remainder of the session, and can be utilised however it is needed. If you have more than one data source that uses the connection, you will need to use the *UpdateConnectionString* function to modify each one in turn. So, for example, if you also had a data source named *Users_DS* that connected to the account's *Users* table, you would need to add the following statement after the first one:

```
UpdateConnectionString("Users_DS",Properties);
```

The final Form Load Expression will look something like this:

```
if:(GetQueryStringText("error_code") = 200)
{
        ShowForm("Cancelled"); //This will prevent the remainder of the expression from being executed;
}

UpdateConnectionString("Users_DS",Properties);

if:(strlen(Code) > 1)
{
        AccessToken:=RunStoredProcedure("GetAccessToken","ClientId,RedirectUri,ClientSecret,Code",2);

        Properties:=FormatString("OAuth Access Token={0}","AccessToken");

        UpdateConnectionString("Posts_DS",Properties);
}
```

With that, you've fully configured the KnowledgeKube application and corresponding Facebook application. Feel free to expand this example to include any additional functionality you require.