You can use global events to integrate external user databases, and modify the authentication or authorization process. See the **SecurityEvents** section of the global event reference to learn more about the available options.

To set up custom authentication, implement a handler for the **SecurityEvents.Authentication.Execute** event. You can access the authentication data through the event handler's **AuthenticationEventArgs** parameter, which provides the following properties:

- **UserInfo User** - an object representing the user attempting to sign in. The object is the result of the system's standard authentication check. If the default authentication failed, the User property is *null*.
- **string UserName** - contains the username entered during the sign-in attempt.
- **string Password** - contains the password entered during the sign-in attempt.

Once you complete the external authentication process in the handler's code, assign a matching *UserInfo* object to the *AuthenticationEventArgs* parameter's **User** property. If the authentication failed, set the property to *null*.

Example

The following example demonstrates how to integrate external user authentication by handling security events:

1. Open your Kentico web project in Visual Studio.
2. Create a custom module class.
   - Either add the class into a custom project within the Kentico solution (recommended) or directly into the Kentico web project (into a custom folder under the **CMSApp** project for *web application* installations, into the **App_Code** folder for *web site* installations).
3. Override the module's **OnInit** method and assign a handler to the **SecurityEvents.Authentication.Execute** event:

```
using System.Data;
using System.Linq;
using System.Web;

using CMS;
using CMS.DataEngine;
using CMS.Membership;

// Registers the custom module into the system
[assembly: RegisterModule(typeof(CustomAuthenticationModule))]

public class CustomAuthenticationModule : Module
{
        // Module class constructor, the system registers the module under the
name "CustomAuthentication"
        public CustomAuthenticationModule()
                : base("CustomAuthentication")
        {
        }

        // Contains initialization code that is executed when the application
starts
        protected override void OnInit()
        {
                base.OnInit();

                // Assigns a handler to the SecurityEvents.Authenticate.Execute
event
                // This event occurs when users attempt to sign in on the website
                SecurityEvents.Authenticate.Execute += OnAuthentication;
        }
}
```

4. Define the handler method within the module class:

```
private void OnAuthentication(object sender, AuthenticationEventArgs e)
{
        // Checks if the user was authenticated by the default system. Only
continues if authentication failed.
        if (e.User == null)
        {
                // Object representing the external user
                UserInfo externalUser = null;

                // Gets the credentials entered during the authentication
                string username = SqlHelper.EscapeQuotes(e.UserName);
                string password = SqlHelper.EscapeQuotes(e.Password);

                // Path to an XML database file
                string xmlPath = HttpContext.Current.Server.MapPath("~
/userdatabase.xml");

                // Reads data from the external database
                DataSet userData = new DataSet();
                userData.ReadXml(xmlPath);

                // Authenticates against the external database
                DataRow[] rows = userData.Tables[0].Select("UserName = '" +
username + "' AND Password='" + password + "'");
                if (rows.Count() > 0)
                {
                        // Creates a user record if external authentication is
successful
                        externalUser = new UserInfo()
                        {
                                IsExternal = true,
                                UserName = e.UserName,
                                FullName = "ExternalUser Fullname",
                                Enabled = true
                        };
                }

                // Passes the object representing the user (or null if external
authentication failed)
                e.User = externalUser;
        }
}
```

5. Save the class.

The system now performs authentication according to user data from the external source if default authentication fails.

We recommend importing all external roles into the **CMS_Role** table of the Kentico database. You can then configure the appropriate permissions for these roles, and fully use the built-in security model together with external users.

If you need to implement custom security logic, create handlers for the available SecurityEvents. You can programmatically check if a user belongs to a role by calling the **IsInRole(string roleName, string siteName)** method for **UserInfo** objects.