Customizing the email provider allows you to:

- Execute custom actions when sending emails (for example logging the sent emails for auditing purposes)
- Use thirdparty components for sending emails

Once you create and register your custom email provider, it is used to process all emails sent out by Kentico.

The following example demonstrates how to define a custom email provider. This sample provider supports all of the default email functionality, but additionally creates an entry in the Event log whenever the system sends an email.

Writing the custom email provider

Start by preparing a separate project for custom classes in your Kentico solution:

1. Open your Kentico solution in Visual Studio.
2. Create a new *Class Library* project in the Kentico solution (or reuse an existing custom project).
3. Add references to the required Kentico libraries (DLLs) for the new project:
    a. Right-click the project and select **Add -> Reference**.
    b. Select the **Browse** tab of the **Reference manager** dialog, click **Browse** and navigate to the ***Lib*** folder of your Kentico web project.
    c. Add references to the following libraries (and any others that you may need in your custom code):
        - **CMS.Base.dll**
        - **CMS.Core.dll**
        - **CMS.DataEngine.dll**
        - **CMS.EmailEngine.dll**
        - **CMS.EventLog.dll**
4. Reference the custom project from the Kentico web project *(CMSApp* or *CMS)*.
5. Edit the custom project's **AssemblyInfo.cs** file (in the *Properties* folder).
6. Add the **AssemblyDiscoverable** assembly attribute:

```
using CMS;

[assembly:AssemblyDiscoverable]
```

Continue by implementing the custom provider class:

1. Add a new class under the custom project. For example, name the class ***CustomEmailProvider.cs***
2. Modify the class's declaration so that it inherits from the **CMS.EmailEngine.EmailProvider** class.

> Custom email providers must always inherit from the default **EmailProvider** class. This allows you to call the base methods before adding your own custom functionality.

3. Register the custom provider by adding the **RegisterCustomProvider** assembly attribute above the class declaration (requires a reference to the **CMS** namespace). The attribute's parameter specifies the type of the custom provider class as a **System.Type** object.
4. Override the following methods in the class:

```
using System.ComponentModel;
using System.Net.Mail;

using CMS;
using CMS.EmailEngine;
using CMS.EventLog;

// Registers the custom EmailProvider
[assembly: RegisterCustomProvider(typeof(CustomEmailProvider))]

public class CustomEmailProvider : EmailProvider
{
    /// <summary>
    /// Synchronously sends an email through the SMTP server.
    /// </summary>
    /// <param name="siteName">Site name</param>
    /// <param name="message">Email message</param>
    /// <param name="smtpServer">SMTP server</param>
    protected override void SendEmailInternal(string siteName, MailMessage
message, SMTPServerInfo smtpServer)
    {
        base.SendEmailInternal(siteName, message, smtpServer);

        string detail = string.Format("Email from {0} was sent via {1}
(synchronously)", message.From.Address, smtpServer.ServerName);
        EventLogProvider.LogInformation("CMSCustom", "EMAIL SENDOUT", detail);
    }

    /// <summary>
    /// Asynchronously sends an email through the SMTP server.
    /// </summary>
    /// <param name="siteName">Site name</param>
    /// <param name="message">Email message</param>
    /// <param name="smtpServer">SMTP server</param>
    /// <param name="emailToken">Email token that represents the message being
sent</param>
    protected override void SendEmailAsyncInternal(string siteName, MailMessage
message, SMTPServerInfo smtpServer, EmailToken emailToken)
    {
        base.SendEmailAsyncInternal(siteName, message, smtpServer, emailToken);

        string detail = string.Format("Email from {0} was dispatched via {1}
(asynchronously)", message.From.Address, smtpServer.ServerName);
        EventLogProvider.LogInformation("CMSCustom", "EMAIL SENDOUT", detail);
    }

    /// <summary>
    /// Raises the SendCompleted event after the send is completed.
    /// </summary>
    /// <param name="e">Provides data for the async SendCompleted event</param>
    protected override void OnSendCompleted(AsyncCompletedEventArgs e)
    {
        base.OnSendCompleted(e);
        string detail = "Received callback from asynchronous dispatch";
        EventLogProvider.LogInformation("CMSCustom", "SEND COMPLETE", detail);
    }
}
```

5. Save all changes and **Build** the custom project.

The provider uses the customized methods to send out emails. Each method calls the base from the parent provider class and then logs an information event into the system's event log. You can use the same approach to add any type of custom functionality.

Result

The system now uses the custom email provider (defined in the **CustomEmailProvider** class) when sending emails.

To try out the functionality:

1. Send some emails using Kentico.
    - For example, you can use the **System -> Email** interface to send a test email (synchronously).
2. Check the log in the **Event log** application.

The event log displays an entry for every successfully sent email, identified by the *CMSCustom* **Source** value (two entries for emails sent asynchronously).