Global events allow you to execute custom code whenever specific actions occur within the system. Events can be raised as a result of both user interaction and the logic of the application itself (default or custom). When you assign a method as a handler for an event, the system automatically executes the code whenever the corresponding action occurs.

For example, when a new page is added to your website, you can use an event handler to load the page's data, send out the content by email, and use a third-party component to generate a PDF version of the page.

Assigning handlers to events

Use code in the following format to register handlers for global events:

<event class>.<event action>.<event type> += <handler method name>

- **Event class** – event classes are containers of events related to groups of functionality
- **Event action** – represents a specific action that occurs within the system
- **Event type** – determines when exactly the event takes place, typically *Before* or *After* the action. Some actions only have one type: *Execute*

For example:

```
DocumentEvents.Update.After += Document_Update_After;
```

The event handlers provide parameters derived from **EventArgs**, which you can use to access data related to the action that occurred. The exact type of the parameter depends on the event.

For full information about the available event classes, actions, event types and handler parameters, see: Reference - Global system events

You need to register event handlers at the beginning of the application's life cycle (during initialization):

1. Create a custom module class.
2. Override the module's **OnInit** method and assign handler methods to events.

For basic execution of initialization code, you only need to register a "code-only" module through the API. You do not need to create a new module within the **Modules** application in the Kentico administration interface.

Example

The following steps describe how to register methods as global event handlers:

1. Open your Kentico project in Visual Studio (using the **WebSite.sln** or **WebApp.sln** file).
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 handler methods to the required events.

```
using CMS;
using CMS.DataEngine;
using CMS.DocumentEngine;

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

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

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

                // Assigns custom handlers to events
                DocumentEvents.Insert.After += Document_Insert_After;
                DocumentEvents.InsertLink.Before += Document_InsertLink_Before;
        }

        private void Document_Insert_After(object sender, DocumentEventArgs e)
        {
                // Add custom actions here
        }

        private void Document_InsertLink_Before(object sender, DocumentEventArgs e)
        {
                // Add custom actions here
        }
}
```

See the child pages in this chapter for examples of event handling.