

[Event handlers](#) allow you to disable [content staging](#) and [integration bus synchronization](#) for certain pages or objects. The system provides two types of events that you can use:

- [Handling the changes of pages or objects](#)
- [Handling the creation of staging or integration tasks](#)

Excluding content by handling page or object changes

When changes are made to pages and objects (such as creating, editing or deleting), the system logs the change data. [Content staging](#) and the [integration bus](#) use the data to create synchronization tasks, which then transfer the changes to the target server or application.

To completely prevent the system from starting the staging or integration process, handle the following events:

- **DocumentEvents.LogChange.Before**
- **ObjectEvents.LogChange.Before** (or **<name>Info.TYPEINFO.Events** for specific object types)

You can use two different approaches to disable staging or integration inside the event handlers:

i Set the properties of the event handler's **LogDocumentChangeArgs/LogObjectChangeArgs** parameter:

```
e.Settings.LogStaging = false;  
e.Settings.LogIntegration = false;
```

Directly switches off logging of staging or integration tasks. The properties can be changed later during the execution of the *LogChange* event.

OR

i Call the **Using** method of the event handler's **LogDocumentChangeArgs/LogObjectChangeArgs** parameter with a new **CMSActionContext** object as the parameter. Set the **LogSynchronization** or **LogIntegration** properties in the **CMSActionContext** constructor.

```
e.Using(new CMSActionContext() {LogSynchronization = false, LogIntegration =  
false} );
```

Disables logging of staging or integration tasks within the entire context of the *LogChange* event.

Example

The following example demonstrates how to use **LogChange** handlers to disable content staging and integration for the */Archive* section of a website.

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 handlers to the **DocumentEvents.LogChange.Before** and **PageTemplateInfo.TYPEINFO.Events.LogChange.Before** events.



```
using CMS;
using CMS.Base;
using CMS.DataEngine;
using CMS.DocumentEngine;
using CMS.PortalEngine;
using CMS.Helpers;

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

public class LogChangeHandlerModule : Module
{
    // Module class constructor, the system registers the module under the
    name "LogChangeHandlers"
    public LogChangeHandlerModule()
        : base("LogChangeHandlers")
    {
        // Contains initialization code that is executed when the application
        starts
        protected override void OnInit()
        {
            base.OnInit();

            // Assigns a handler to the DocumentEvents.LogChange.Before event
            // This event occurs when pages are modified, before the system
            starts logging the changes
            DocumentEvents.LogChange.Before += LogDocumentChange_Before;

            // Assigns a handler to the LogChange.Before event for
            PageTemplateInfo objects
            // This event occurs when page templates are modified, before the
            system starts logging the changes
            PageTemplateInfo.TYPEINFO.Events.LogChange.Before +=
            LogPageTemplateObjectChange_Before;
        }
    }
}
```

4. Define the handler methods (inside the custom module class):



```
// Disables logging of staging and outgoing integration tasks for the content of
pages in the /Archive section of the website
private void LogDocumentChange_Before(object sender, LogDocumentChangeEventArgs e)
{
    // Gets the synchronized page
    TreeNode page = e.Settings.Node;

    // Excludes pages under the "/Archive" path from content staging and
integration bus synchronization
    if (page.NodeAliasPath.StartsWith("/Archive"))
    {
        e.Using(new CMSActionContext()
        {
            LogSynchronization = false,
            LogIntegration = false
        });
    }
}

// Disables 'Create' and 'Update' staging/integration tasks for ad-hoc page
templates of pages in the excluded /Archive section
private void LogPageTemplateObjectChange_Before(object sender,
LogObjectChangeEventArgs e)
{
    // Gets the synchronized object
    GeneralizedInfo obj = e.Settings.InfoObj;

    if (obj != null)
    {
        // Continues only for ad-hoc templates
        if (!ValidationHelper.GetBoolean(obj.GetValue(
("PageTemplateIsReusable"), false))
        {
            // Checks if the template is used by a page in the
/Archive section
            var nodeQuery = DocumentHelper.GetDocuments()

.Path("/Archive", PathTypeEnum.Section)

.WhereEquals("NodeTemplateID", obj.GetValue("PageTemplateID"));

            if (nodeQuery.Count > 0)
            {
                // Disables logging of staging and integration
tasks
                e.Settings.LogStaging = false;
                e.Settings.LogIntegration = false;
            }
        }
    }
}
```

5. Save the class.

The custom handlers ensure that the system does not create synchronization tasks for:





- The content and metadata of pages in the website's */Archive* section
- [Ad-hoc page templates](#) of the pages in the */Archive* section (only *Create* and *Update* synchronization tasks, does not disable *Delete* tasks)

Excluding content by handling synchronization tasks

[Content staging](#) and the [integration bus](#) use synchronization tasks to transfer changes to the target server or application. You can cancel the creation of synchronization tasks during the following system events:

- **StagingEvents.LogTask.Before**
- **IntegrationEvents.LogInternalTask.Before**

Excluding content via the *LogTask* events provides the following advantages and disadvantages:

-  Allows you to access the data of the synchronization task inside the event handler. For example, you can disable only specific types of synchronization tasks.
-  The system needs to prepare the synchronization data for every object/page change, which requires more overhead than when excluding content through [LogChange handlers](#).

Example

The following example demonstrates how to use the **LogTask** handler to disable content staging for the deletion of [role objects](#) and for content synchronization of certain pages.

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 a handler to the **StagingEvents.LogTask.Before** event.



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

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

public class LogTaskHandlerModule : Module
{
    // Module class constructor, the system registers the module under the
    name "LogTaskHandlers"
    public LogTaskHandlerModule()
        : base("LogTaskHandlers")
    {
        // Contains initialization code that is executed when the application
        starts
        protected override void OnInit()
        {
            base.OnInit();

            // Assigns a handler to the StagingEvents.LogTask.Before event
            // This event occurs before the system creates content staging
            synchronization tasks
            StagingEvents.LogTask.Before += LogTask_Before;
        }
    }
}
```

4. Define the handler method (inside the custom module class):



```
private void LogTask_Before(object sender, StagingLogTaskEventArgs e)
{
    // Handles content staging exclusion of documents
    if (e.Object is TreeNode)
    {
        // Gets the synchronized document
        TreeNode document = (TreeNode)e.Object;

        // Cancels the creation of content staging tasks for documents
        under the "/Archive" path
        if (document.NodeAliasPath.StartsWith("/Archive"))
        {
            e.Cancel();
        }
    }
    // Handles content staging exclusion of objects
    else
    {
        // Gets the synchronized object
        BaseInfo synchronizedObject = e.Object;

        // Gets the synchronization task
        StagingTaskInfo stagingTask = e.Task;

        // Cancels the creation of content staging tasks for the deletion
        of role objects
        if ((synchronizedObject.TypeInfo.ObjectType == CMS.Membership.
        RoleInfo.OBJECT_TYPE)
            && (stagingTask.TaskType == TaskTypeEnum.DeleteObject))
        {
            e.Cancel();
        }
    }
}
```

5. Save the class.

The handler ensures that the system does not create staging tasks:

- For changes made to pages in the website's */Archive* section
- When a role is deleted from the system