When using staging to synchronize objects between Kentico instances, the system does not generate distinct staging tasks for most types of child objects and bindings. Child objects and bindings are often included within the data of staging tasks related to their parent object.

Staging of child and binding data may cause problems because different environments require different types of child data processing. In some cases, you may wish to synchronize the exact state of objects from the source server, including deletion of objects that exist only on the target server. Other environments assume that relevant changes occur on the target server and need to avoid deletion of objects that do not exist on the source.

As a result, the default behavior in Kentico may not meet the requirements of your environment. You can change the default staging behavior for specific child or binding object types by creating event handlers for the **StagingEvents.GetChildProcessingType** event.

The *GetChildProcessingType* event occurs on ***target staging servers*** when processing create or update tasks that include the data of child or binding objects, once for each child or binding type. For example, staging tasks for roles include the data of binding objects, such as user-role and role-permission relationships. When processing a create or update staging task for a role, the *GetChildProcessingType* event triggers separately for each binding type included in the role data.

Creating GetChildProcessingType handlers

Use the following process to create handlers for the *GetChildProcessingType* event:

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.GetChildProcessingType.Execute** event.
4. Implement the handler method (inside the custom module class):
   a. Write conditions for specific object types based on the **ObjectType** and **ParentObjectType** properties of the handler's *StagingChildProcessingTypeEventArgs* parameter.
   b. Set the required staging behavior through the **ProcessingType** property of the handler's *StagingChildProcessingTypeEventArgs* parameter.

   > The **ProcessingType** property uses values from the *IncludeToParentEnum* enumeration, with the following possible options:
   >
   > - **Complete** – Staging tasks synchronize objects to exactly match the incoming data. Any objects missing in the task data are ***deleted***.
   > - **Incremental** – Staging tasks only add new objects or update existing ones. Objects missing in the task data are NOT deleted.
   > - **None** – Processing of staging data is disabled for the given child or binding object type and no updates occur.
   >
   > **Note**: The system only triggers the *GetChildProcessingType* event for child objects that support staging as part of their parent object's data. You cannot use the *Complete* or *Incremental* values to enable staging for child or binding objects that are not included in parent data.

5. Transfer the module code to all instances in your staging environment where you wish to apply the customization.

When processing incoming staging tasks, the system handles child and binding object data according to the *ProcessingType* that you set in the *GetChildProcessingType* handler.

Example

The following code demonstrates how to create a custom module that changes the default staging behavior for user-role bindings. The customization switches the **ProcessingType** to *Incremental,* which ensures that incoming staging tasks do not delete existing user-role bindings.

```
using System;

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

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

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

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

                // Assigns a handler to the StagingEvents.GetChildProcessingType.
Execute event
                // This event occurs when the system processes incoming staging tasks,
once for each child or binding object type in the task's data
                StagingEvents.GetChildProcessingType.Execute +=
Staging_GetChildProcessingType;
        }

        private void Staging_GetChildProcessingType(object sender,
StagingChildProcessingTypeEventArgs e)
        {
                // Switches the staging to incremental mode for user-role bindings
                if (String.Equals(e.ParentObjectType, RoleInfo.OBJECT_TYPE) && String.
Equals(e.ObjectType, UserRoleInfo.OBJECT_TYPE))
                {
                        e.ProcessingType = IncludeToParentEnum.Incremental;
                }
        }
}
```