When transferring data between instances of Kentico (via import/export or staging), the IDs of objects can differ between the environments. Kentico automatically "translates" the ID values for system objects, but not for custom objects. For example, when you stage a custom table or page type that contains a custom field storing object IDs, the values may not match the IDs of the objects on the target server.

You can ensure that the system correctly translates custom ID fields by handling the events in the **ColumnsTranslationEvents** class:

- **RegisterRecords** – occurs on the source application when exporting objects. Handle this event to provide information for custom field translation.
- **TranslateColumns** – occurs on the target application when importing objects. Handle this event to translate field values using the provided information.

Important

Handle the *RegisterRecords* event on the ***source*** application, and the *TranslateColumns* event on the ***target*** application. Add the handlers to both applications to ensure correct ID translation for both directions (for example bi-directional staging).

Example

The following example demonstrates how to implement custom ID translation for the *CustomTableUserID* field of a custom table. The field stores the IDs of the users who own individual records in the custom table.

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 **A pp_Code** folder for *web site* installations).
3. Override the module's **OnInit** method and assign a handlers to the **ColumnsTranslationEvents.RegisterRecords.Execute** and **ColumnsTranslationEvents.TranslateColumns.Execute** events.

```
using CMS;
using CMS.DataEngine;
using CMS.CustomTables;
using CMS.Helpers;
using CMS.Membership;

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

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

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

                // Add this code on the SOURCE application from which you are
exporting the objects
                // Assigns a handler method to the RegisterRecords event
                ColumnsTranslationEvents.RegisterRecords.Execute +=
RegisterRecords_Execute;

                // Add this code on the TARGET application where you are
importing the objects
                // Assigns a handler method to the TranslateColumns event
                ColumnsTranslationEvents.TranslateColumns.Execute +=
TranslateColumns_Execute;
        }
}
```

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

```
// Provides the ID translation data when exporting objects
private void RegisterRecords_Execute(object sender, ColumnsTranslationEventArgs e)
{
        // Registers custom column translation data for "customtable.sampletable"
        if (e.ObjectType == CustomTableItemProvider.GetObjectType("customtable.
sampletable"))
        {
                // Gets the user ID from the custom table's data
                int userId = ValidationHelper.GetInteger(e.Data.GetValue
("CustomTableUserID"), 0);

                // Prepares parameters for the translation
                TranslationParameters parameters = new TranslationParameters
(UserInfo.OBJECT_TYPE)
                {
                        CodeName = UserInfoProvider.GetUserNameById(userId)
                };

                // Maps the user ID value to the corresponding username for every
record in the custom table's data
                // Adds the mapping information into the export package
                e.TranslationHelper.RegisterRecord(userId, parameters);
        }
}

// Handles ID translation when importing objects
private void TranslateColumns_Execute(object sender, ColumnsTranslationEventArgs
e)
{
        // Performs column translation for "customtable.sampletable"
        if (e.ObjectType == CustomTableItemProvider.GetObjectType("customtable.
sampletable"))
        {
                // Translates the values of the UserID field in the table's data
                // Uses the mapping information in the import package to find the
correct IDs based on the usernames
                e.TranslationHelper.TranslateColumn(e.Data, "CustomTableUserID",
UserInfo.OBJECT_TYPE, 0, true, true);
        }
}
```

5. Save the class.

When you transfer the custom table's data between the applications using import/export or content staging, the handlers ensure correct translation of ID values for the *CustomTableUserID* field.