Running queries

You can execute system queries in your custom code.

Examples

```
using System.Data;
using CMS.DataEngine;

...

// Executes the cms.user.selectall query, with specified columns, and a WHERE and
ORDER BY clause
DataSet users = new DataQuery("cms.user.selectall")
        .Columns("UserID", "UserName", "FullName")
        .Where("UserName", QueryOperator.Like, "%admin%")
        .OrderBy("FullName")
        .Execute();

// Assigns the value "administrator" to the "@UserName" query parameter
QueryDataParameters parameters = new QueryDataParameters();
parameters.Add("@UserName", "administrator");

// Executes the cms.user.selectbyusername query
// Uses the "administrator" value for the "@UserName" parameter in the query's code
var query = new DataQuery("cms.user.selectbyusername");
query.Parameters = parameters;
DataSet selectedUser = query.Result;
```

To create custom queries or modify existing ones, use one of the following approaches:

- Manually edit the **CMS_Query** database table.
- Manage queries through the administration interface in:
    - **Page types -> Edit page type -> Queries**
    - **Custom tables -> Edit table -> Queries**
    - **Modules -> Edit module -> Classes -> Edit class -> Queries**

Pre-processing queries

You can pre-process database queries using the **ExecuteQuery.Before** event of the **SqlEvents** class. The system raises the event before executing any database query. The event allows you to dynamically modify the behavior and code of queries.

To create a handler for the *ExecuteQuery.Before* event:

1. Open your Kentico web 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).

      
      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.

3. Override the module's **OnInit** method and assign a handler method to the **SqlEvents.ExecuteQuery.Before** event.

4. Define the handler method as required.

The system automatically runs the module's **OnInit** method when the application starts, which registers your event handler.

The following handler example replaces *CMS_User* with *View_CMS_User* in the query code when processing the *cms.user.selectall* query:

```
using System.Data;

using CMS;
using CMS.DataEngine;

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

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

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

                SqlEvents.ExecuteQuery.Before += BeforeExecuteQuery;
        }

        // Replaces CMS_User with View_CMS_User in the query code when processing the
cms.user.selectall query
        static void BeforeExecuteQuery(object sender, ExecuteQueryEventArgs<DataSet> e)
        {
                if (e.Query.Name != null)
                {
                        switch (e.Query.Name.ToLower())
                        {
                                case "cms.user.selectall":
                                        e.Query.Text = e.Query.Text.Replace
("CMS_User", "View_CMS_User");
                                        break;
                        }
                }
        }
}
```

Post-processing queries

You can process the results of queries using the **ExecuteQuery.After** event of the **SqlEvents** class. The system raises the event after executing any database query. The event allows you to use or modify the data retrieved by queries.

To create a handler for the *ExecuteQuery.After* event:

1. Open your Kentico web 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).

  > 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.

3. Override the module's **OnInit** method and assign a handler method to the **SqlEvents.ExecuteQuery.After** event.
4. Define the handler method as required.

The system automatically runs the module's **OnInit** method when the application starts, which registers your event handler.

The following handler example dynamically generates the full name of users and overrides the default full name (whenever the *cms.user.selectall* query is executed).

```
using System.Data;

using CMS;
using CMS.DataEngine;

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

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

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

                SqlEvents.ExecuteQuery.After += AfterExecuteQuery;
        }

        // Generates the full name of users and overrides the default full name
whenever the cms.user.selectall query is executed
        static void AfterExecuteQuery(object sender, ExecuteQueryEventArgs<DataSet> e)
        {
                if (e.Query.Name != null)
                {
                        switch (e.Query.Name.ToLower())
                        {
                                case "cms.user.selectall":
                                        if (e.Result != null)
                                        {
                                                DataTable dt = e.Result.Tables[0];
                                                foreach (DataRow dr in dt.Rows)
                                                {
                                                        dr["FullName"] = dr
["FirstName"] + " " + dr["MiddleName"] + " " + dr["LastName"];
                                                }
                                        }
                                        break;
                        }
                }
        }
}
```