Creatio Academy

Interface control tools

Version 8.0

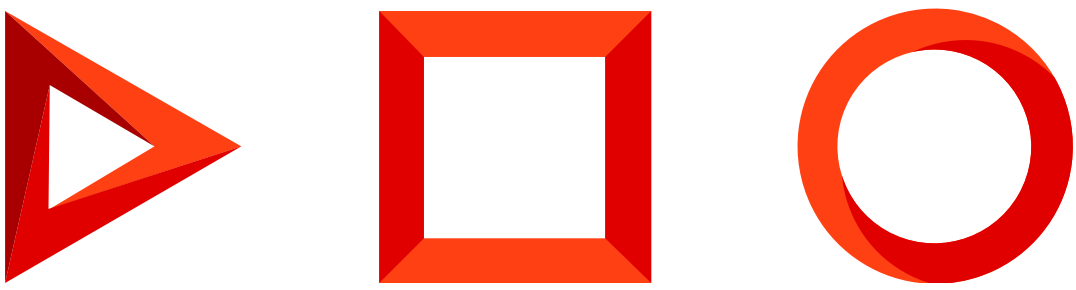


Table of Contents

Manage an existing additional feature

Advanced

This article covers the additional feature behavior relevant to Creatio 8.0.2 Atlas and later. If you use an earlier Creatio version, follow the guide in a separate article: Feature Toggle mechanism.

Feature toggle is a software development technique. The **purpose** of the feature toggle is to manage the additional feature status in a live application.

Feature toggle lets you use continuous integration while preserving the working capacity of the application and hiding features you are still developing. Learn more about developing custom features in a separate article: Implement a custom additional feature. Use the [*Feature toggling*] page to manage the additional feature status.

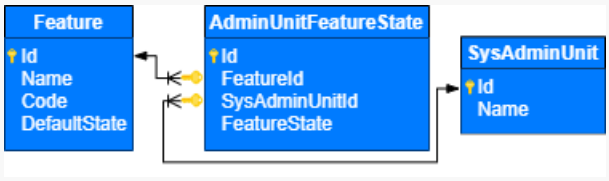
Learn more about the Feature toggle mechanism in Wikipedia.

Store the additional feature details

Creatio stores the additional feature details in the following database **tables**:

- `[Feature]`. Stores the list of features you can toggle. Empty by default.
- `[AdminUnitFeatureState]` (the `[FeatureState]` column). Stores the additional feature status (turned on/off). The `[AdminUnitFeatureState]` table connects the `[Feature]` and the `[SysAdminUnit]` database tables. `[SysAdminUnit]` is the additional feature status for Creatio users and user groups.

View the relationship chart between tables that store the additional feature details in the figure below.



View the primary columns of the `[Feature]` database table below.

Primary columns of the `[Feature]` database table

Column	Type	Description
`[Id]`	`uniqueidentifier`	The unique ID of the record.
`[Name]`	`nvarchar(250)`	The name of the additional feature. By default, matches the `[Code]` column value. You can change the name of the additional feature manually.
`[Code]`	`nvarchar( 50 )`	The code of the additional feature.
`[DefaultState]`	`int`	The default status of the additional feature (`1` : turned on, `0` : turned off).

View the primary columns of the `[AdminUnitFeatureState]` database table below.

Primary columns of the `[AdminUnitFeatureState]` database table

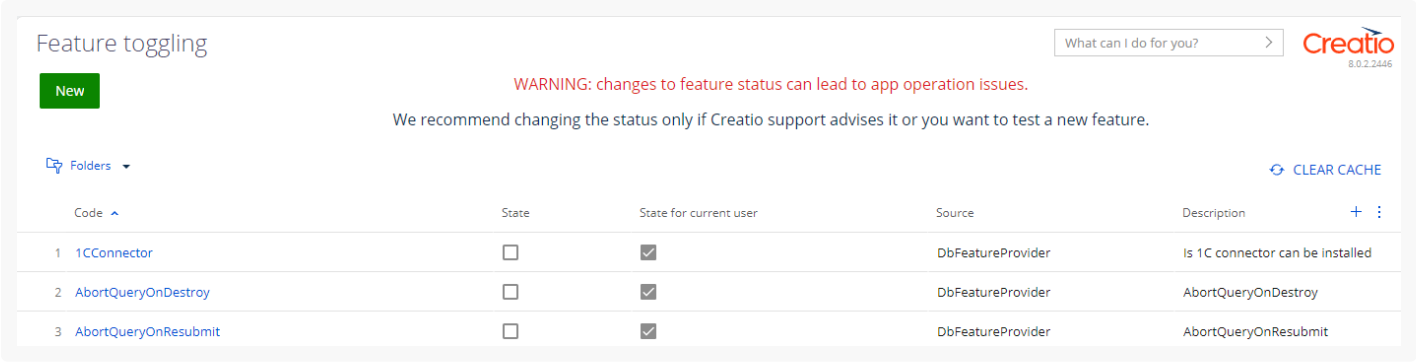
Column	Type	Description
`[Id]`	`uniqueidentifier`	The unique ID of the record.
`[FeatureId]`	`uniqueidentifier`	The unique ID of the record from the `[Feature]` database table.
`[SysAdminUnitId]`	`uniqueidentifier`	The unique ID of the user record.
`[FeatureState]`	`int`	The status of the additional feature (`1` : turned on, `0` : turned off).

Open the [*Feature toggling*] page

You can open the [*Feature toggling*] page in several **ways**:

- Use the `[Creatio URL]/0/Features` link. For example, `http://my.creatio.com/0/Features`.
- Use the `[Creatio URL]/0/Flags` link. For example, `http://my.creatio.com/0/Flags`.

As a result, the browser will open the [*Feature toggling*] page.



The [*Feature toggling*] page displays the following **data**:

- additional feature code (the [*Code*] column)
- additional feature status for all users (the [*State*] column)
- additional feature status for the current user (the [*State for current user*] column)
- additional feature source (the [*Source*] column)
- additional feature description (the [*Description*] column)

The additional feature source (the [*Source*] column) can have the following **types**:

- `DbFeatureProvider`. Records of the `[Feature]` database table.
- `web.config`. Boolean key values from the `<appSettings>` section of the `Web.config` file in Creatio root directory.
- `Metadata`. Additional features registered in the source code using the class that inherits from `FeatureMetadata` .
- `GlobalAppSettings`. Boolean properties of the `Terrasoft.Core.GlobalAppSettings` class. Learn more in the .NET class library.

The types of additional feature source are sorted by priority in descending order.

The **rules** that determine the additional feature status are as follows:

- If an additional feature is not registered in Creatio, it is disabled.
- If an additional feature status is specified in multiple sources, the resulting status is based on the source priority. Back-end changes to the additional feature status do not require you to refresh the browser page. Refresh the page to apply the front-end changes to the additional feature status.

View the examples of the additional feature status (`true` : turned on, `false` : turned off) determined based on multiple sources below.

Examples of the additional feature status definition

Feature source			Resulting feature status
`DbFeatureProvider`	`web.config`	`Metadata` or `GlobalAppSettings`	
`true`	`false`	`false`	`true`
`false`	`true`	`false`	`false`
Not available	`true`	`false`	`true`
Not available	`false`	`true`	`false`

Manage the additional feature status

Attention. Changes to the additional feature status can affect Creatio operability. We recommend changing the feature status only if support team approves it or you need to teat a custom feature.

You can manage the additional feature status in several **ways**:

- for a Creatio user group
- for all Creatio users

Since version 8.0.2 Atlas, Creatio lets you manage the additional feature status using the WorkspaceConsole utility. To do this, follow the guide in a separate article: Delivery in WorkspaceConsole.

Change the additional feature status for user groups

Since version 8.0.2 Atlas, Creatio lets you change the additional feature status for user groups.

To change the additional feature status **for user groups**:

1. Open the [*Feature toggling*] page.
2. Open the page of the additional feature whose status to change for the user group.
3. If the group is absent from the [*Feature status for the system administration object*] detail, add it.
   a. Click the + button on the [*Feature status for the system administration object*] detail.
   b. Select the corresponding user group in the [*Admin unit*] field.
4. Open the user group page.
5. Select or clean the [*Feature state*] checkbox.
6. Click [*Save*].
7. Refresh the additional feature page to apply the changes.

Attention. If the additional feature is turned on for all users and turned off for a user group, Creatio disables it only for the user group.

Change the additional feature status for all users

To change the additional feature status **for all users**:

1. Open the [*Feature toggling*] page.
2. Open the page of the additional feature whose status to change for all users.
3. Select or clean the [*State*] checkbox.
4. Click [*Save*].

FeatureUtilities class C#

Medium

The `Terrasoft.Configuration` namespace.

The `Terrasoft.Configuration.FeatureUtilities` class provides a set of extension methods to the `UserConnection` class. These methods let you use the `Feature Toggle` functionality in the source code schemas of Creatio back-end. The `FeatureUtilities` class also declares the enumeration of feature statuses (the `[FeatureState]` column of the `[AdminUnitFeatureState]` database table).

Note. Use the .NET class reference to access the full list of the constructors, methods and properties, parent classes, and implemented interfaces of the `UserConnection` class.

Methods

`static int GetFeatureState(this UserConnection source, string code)`

Returns the feature status.

Parameters

`source`	The user connection.
`code`	The feature code.

`static int GetFeatureState(this UserConnection source, string code, Guid sysAdminUnitId)`

Returns the feature status.

Parameters

source	The user connection.
code	The code of the feature.
sysAdminUnitId	The unique record ID.

`static bool GetIsFeatureEnabledForAnyUser(this UserConnection source, string code)`

Returns the feature status for any user.

Parameters

source	The user connection.
code	The code of the feature.

`static bool GetIsFeatureEnabledForAllUsers(this UserConnection source, string code)`

Returns the feature status for all users.

Parameters

source	The user connection.
code	The code of the feature.

`static Dictionary<string, int> GetFeatureStates(this UserConnection source)`

Returns all feature statuses.

Parameters

source	The user connection.

`static List<FeatureInfo> GetFeaturesInfo(this UserConnection source)`

Returns data about all features and their statuses.

Parameters

source	The user connection.

```
static void SetFeatureState(this UserConnection source, string code, int state, bool forAllUsers
```

Sets the feature status.

Parameters

source	The user connection.
code	The code of the feature.
state	The new status of the feature.
forAllUsers	Sets the status of the feature for all users.

```
static void SafeSetFeatureState(this UserConnection source, string code, int state, bool forAllU
```

Sets the feature status or creates the feature if it does not exist.

Parameters

source	The user connection.
code	The code of the feature.
state	The new status of the feature.
forAllUsers	Sets the status of the feature for all users.

```
static void CreateFeature(this UserConnection source, string code, string name, string descripti
```

Creates the feature.

Parameters

`source`	The user connection.
`code`	The code of the feature.
`name`	The name of the feature.
`description`	The description of the feature.

```
static bool GetIsFeatureEnabled(this UserConnection source, string code)
```

Checks if the feature is turned on.

Parameters

`source`	The user connection.
`code`	The code of the feature.

```
static bool GetIsFeatureEnabled(this UserConnection source, string code, Guid sysAdminUnitId)
```

Checks if the feature is turned off.

Parameters

`source`	The user connection.
`code`	The code of the feature.
`sysAdminUnitId`	The unique record ID.

`FeatureState` enumeration

The `FeatureState` enumeration defines the feature statuses (the `[FeatureState]` column of the `[AdminUnitFeatureState]` database table).

`Disabled`	`0`	The feature is turned off.
`Enabled`	`1`	The feature is turned on.
`Established`	`2`	The feature is established.

Implement a custom additional feature

Medium

This article covers the additional feature behavior relevant for Creatio version 8.0.2 Atlas and later. If you use an earlier Creatio version, follow the guide in a separate article: Feature Toggle mechanism.

Feature toggle is a software development technique. The **purpose** of the feature toggle is to manage the additional feature status in a live application.

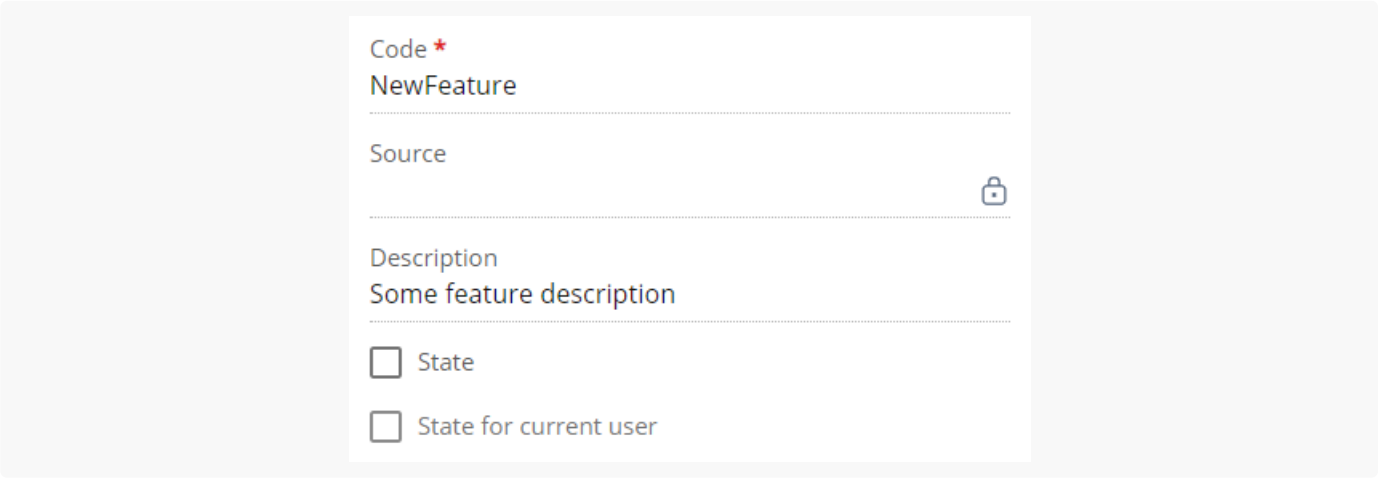
Feature toggle lets you use continuous integration while preserving the working capacity of the application and hiding features you are still developing. Learn more about developing custom features in a separate article: Manage an existing additional feature. Use the [*Feature toggling*] page to manage the additional feature status.

To **implement a custom additional feature**:

1. Add a custom feature.
2. Register the custom feature.
3. Implement the business logic of the custom feature.
4. Set up the custom feature status for Creatio users.

1. Add a feature

1. Open the [*Feature toggling*] page.
2. Click the [*Add*] button on the page toolbar and fill out the **properties of the feature to add**:
   - [*Code*] is the code of the custom additional feature to add. Required.
   - [*Source*] is the source of the custom feature to add. By default, Creatio adds the feature to the `[Feature]` database table.
   - [*Description*] is the description of the custom feature to add.

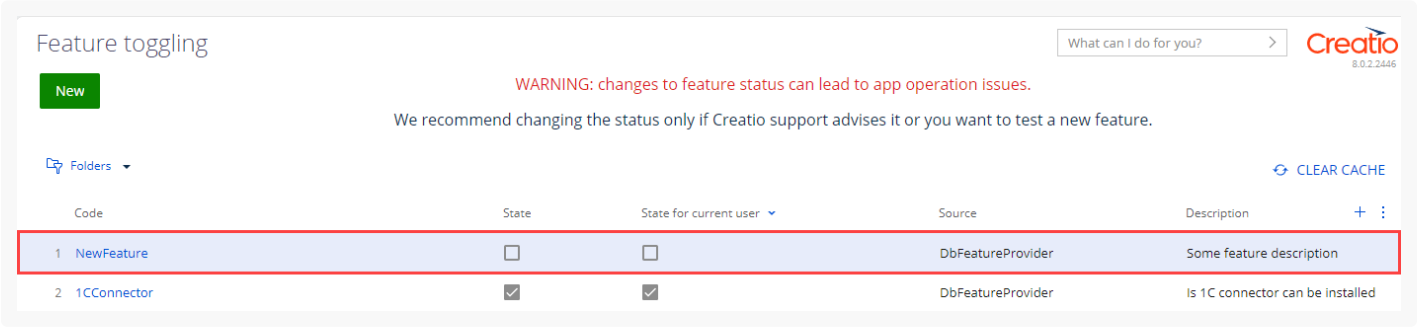


3. Turn on the additional feature.
   - To change the additional feature status for all users, follow the guide in a separate article: Manage an existing additional feature.
   - To change the additional feature status for user groups, follow the guide in a separate article: Manage an

existing additional feature.

4. Click [*Save*].

As a result, Creatio will add the custom feature to the [*Feature toggling*] page list. The source of the feature will be set to `DbFeatureProvider`. Learn more about feature sources in a separate article: Manage an existing additional feature.



2. Register the feature

Register the feature in the back-end using a [*Source code*] schema.

To **register a feature**:

1. Create a [*Source code*] schema. To do this, follow the guide in a separate article: Source code (C#).
2. Register the custom feature in the Source Code Designer. To do this, implement a custom class that inherits from the `FeatureMetadata` class. You can use the template below.

Template to register a custom feature

```
#region Class: SomeNewFeature
internal class SomeNewFeature : FeatureMetadata {

    #region Constructors: Public
    public SomeNewFeature() {
        IsEnabled = true;
        Description = "Some feature description";
    }
    #endregion

}
#endregion
```

If you want to implement multiple features, use a grouping class. In this class, register a custom class for each feature. This lets you reduce the time required to implement the mechanism that retrieves the feature statuses. You can use the template below.

Template to register a custom feature collection

```
class SomeModuleFeatures {
    class SomeNewFeature1 : FeatureMetadata {}
    class SomeNewFeature2 : FeatureMetadata {}
}
Features.GetIsEnabled<SomeModuleFeatures.SomeNewFeature1>()
```

3. Click [*Save*] on the Source Code Designer's toolbar to save the changes to Creatio metadata temporarily.
4. Click [*Publish*] on the Source Code Designer's toolbar to apply the changes to the database level.

3. Implement the business logic of the feature

Implement the custom feature in the block of the conditional operator that checks the feature status (i. e. the `[FeatureState]` column value from the `[AdminUnitFeatureState]` database table).

You can implement the business logic of the custom feature in the following **ways**:

- In the front-end. To do this, follow the guide in a different section: Implement a business logic of a feature in the front-end.
- In the back-end. To do this, follow the guide in a different section: Implement a business logic of a feature in the back-end.

Implement the business logic of the feature in the front-end

1. Create a view model schema. To do this, follow the guide in a separate article: Client module.
2. Add the source code in the Module Designer. The source code of the module schema contains the additional feature block and the conditional operator that checks the feature status and specifies Creatio behavior for each status. You can use the template below.

Template to implement a custom feature

```
/* Method that defines the feature. */
someMethod: function() {

    /* Check the custom feature status. */
    if (Terrasoft.Features.getIsEnabled("SomeNewFeature")) {
        /* Implement the business logic to execute, if the custom feature is turned on. */
        ...
    }

    /* Check the custom feature status. */
    if (Terrasoft.Features.getIsDisabled("SomeNewFeature")) {
        /* Implement the business logic to execute, if the custom feature is turned off. */
        ...
    }
```

```
    /* Implement the method. */
    ...
}
```

The `getIsEnabled()` method in the front-end checks if the custom feature is turned on. The feature name is `SomeNewFeature` in the template above.

The `getIsDisabled()` method in the front-end checks if the custom feature is turned off. The feature name is `SomeNewFeature` in the template above.

The business logic to execute depends on the retrieved value.

3. Click [*Save*] on the Module Designer's toolbar.

Refresh the page to add the custom feature to the client code and display the feature on the browser page.

Implement the business logic of the feature in the back-end

1. Create a [*Source code*] schema. To do this, follow the guide in a separate article: Source code (C#).
2. Implement the custom feature in the Source Code Designer. The application source code contains the additional feature block and the conditional operator that checks the feature status and specifies Creatio behavior for each status.

   The `Terrasoft.Configuration.FeatureUtilities` class provides a set of extension methods to the `UserConnection` class. These methods let you use the `Feature toggle` functionality in the source code schemas of the Creatio back-end.You can use the template below.

Template to implement a custom feature

```
/* The namespace that lets you toggle a feature. */
using Terrasoft.Configuration;
...
/* The method to implement a feature. */
public void AnyMethod() {

    /* Check the custom feature status. */
    if (Features.GetIsEnabled("SomeNewFeature")) {
        /* Implement the business logic to execute if the custom feature is turned on. */
        ...
    }

    /* Check the custom feature status. */
    if (Features.GetIsDisabled("SomeNewFeature")) {
        /* Implement the business logic to execute if the custom feature is turned off. */
        ...
    }

    /* Implement the method. */
    ...
```

```
}
```

The `GetIsEnabled()` method in the back-end checks if the custom feature is turned on. The feature name is `SomeNewFeature` in the template above.

The `GetIsDisabled()` method in the back-end checks if the custom feature is turned off. The feature name is `SomeNewFeature` in the template above.

The business logic to execute depends on the retrieved value.

If you use a grouping class, implement a custom feature collection using the template below.

Template to implement a custom feature

```
/* The namespace that lets you toggle a feature. */
using Terrasoft.Configuration;
...
/* The method to implement a feature. */
public void AnyMethod() {

    /* Check the custom feature statuses that are registered in the class. */
    if (Features.GetIsEnabled<SomeModuleFeatures>()) {
        /* Implement the business logic to execute if the custom feature is turned on. */
        ...
    }

    /* Check the custom feature statuses that are registered in the class. */
    if (Features.GetIsDisabled<SomeModuleFeatures>()) {
        /* Implement the business logic to execute if the custom feature is turned off. */
        ...
    }

    /* Implement the method. */
    ...
}
```

The `GetIsEnabled()` method in the back-end checks if the custom features that are registered in the grouping class are turned on. The class name is `SomeModuleFeatures` in the template above.

The `GetIsDisabled()` method in the back-end checks if the custom features that are registered in the grouping class are turned off. The class name is `SomeModuleFeatures` in the template above.

The business logic to execute depends on the retrieved value.

3. Click [*Save*] on the Source Code Designer's toolbar to save the changes to Creatio metadata temporarily.
4. Click [*Publish*] on the Source Code Designer's toolbar to apply the changes to the database level.

4. Set up the feature status for Creatio users

To **set up the custom feature status for Creatio users**, follow the guide in a separate article: Manage an existing additional feature.

Locking edit page fields

Advanced

During the development of the Creatio custom functions, you may need to lock all fields and details on an edit page under certain conditions. The mechanism of locking edit page fields lets you implement page business logic without creating additional business rules.

Attention. The locking mechanism is available in version 7.11.1 and up.

Attention. You can disable the locking of edit page fields using the `CompleteCardLockout` option on the Feature Toggle page . The Feature Toggle page is available via the following URL: `../0/Nui/ViewModule.aspx#BaseSchemaModuleV2/FeaturesPage` . For example, `https://myserver.com/CreatioWebApp/0/Nui/ViewModule.aspx#BaseSchemaModuleV2/FeaturesPage`

As a result of applying the locking mechanism on an edit page, all fields and details will become locked. If the field has a binding for the `enabled` property in the `diff` array element or the business rule, the mechanism will not lock this field. Locking a detail will hide buttons and menu items for performing operations with the detail records. A locked detail with an editable list will still feature an ability to access the object page, however, all fields on it will be locked.

Attention. The locking mechanism is intended for locking details with a regular list and an editable list. To ensure the correct operation of the mechanism for details with editable fields, create a replacement schema for this detail and control the availability of fields using the `IsEnabled` attribute.

To enable the locking mechanism, set the source code of the edit page to `false` for the `IsModelItemsEnabled` model attribute:

```
this.set(“IsModelItemsEnabled”, false);
```

Alternatively, set the default value for the attribute:

```
"IsModelItemsEnabled": {
 dataValueType: Terrasoft.DataValueType.BOOLEAN,
 value: true,
 dependencies: [{
  columns: ["PaymentStatus"],
  methodName: "setCardLockoutStatus"
```

```
 }]
}
```

Additionally, to operate the locking mechanism on a specific edit page in the `diff` array of this page, specify the `DisableControlsGenerator` generator for the containers in which you want to lock fields. Therefore, to lock all fields of an edit page, specify the global `CardContentWrapper` container:

```
diff: /**SCHEMA_DIFF*/[
 {
  "operation": "merge",
  "name": "CardContentWrapper",
  "values": {
   "generator": "DisableControlsGenerator.generatePartial"
  }
 }
]/**SCHEMA_DIFF*/
```

Attention. In versions 7.13.0 and below, an attempt to specify a generator value will trigger an error when opening the edit page in the Section Wizard.

To fix the error:

1. Find out the name of the group containing all employees. To do this, use the following DB query.

   ```
   select Name from SysAdminUnit
   ```

2. Run the script provided below on your database. Note that the `@allEmpoyeeGroupName` field must be filled in with the name of the group containing all employees of your organization.

   ```
   DECLARE @allEmpoyeeGroupName nvarchar(max) = 'All employees';
   DECLARE @featureName nvarchar(max) = 'PageDesignerCustomGeneratorFix'
   DECLARE @featureStatus bit = 1;

   IF (NOT EXISTS (SELECT NULL FROM Feature WHERE Code = @featureName))
   BEGIN
     INSERT INTO Feature (Name, Description, Code, ProcessListeners)
     VALUES (@featureName, @featureName, @featureName, 0)
   END
   IF(EXISTS (SELECT NULL FROM AdminUnitFeatureState
            WHERE FeatureId = (SELECT Id FROM Feature WHERE Code = @featureName) AND
              SysAdminUnitId = (SELECT Id FROM SysAdminUnit WHERE Name = @allEmpoyeeGroupName))
   BEGIN
   ```

```
  UPDATE AdminUnitFeatureState SET FeatureState = @featureStatus WHERE FeatureId = (SELECT Id
                              SysAdminUnitId = (SELECT Id FROM SysAdminUnit WHERE Name = @all
END
ELSE
BEGIN
  INSERT INTO AdminUnitFeatureState (ProcessListeners, SysAdminUnitId, FeatureState, FeatureI
                    (
                    0,
                    (SELECT Id FROM SysAdminUnit WHERE Name = @allEmpoyeeGroupName),
                    @featureStatus,
                    (SELECT Id FROM Feature WHERE Code = @featureName)
                     )
END
```

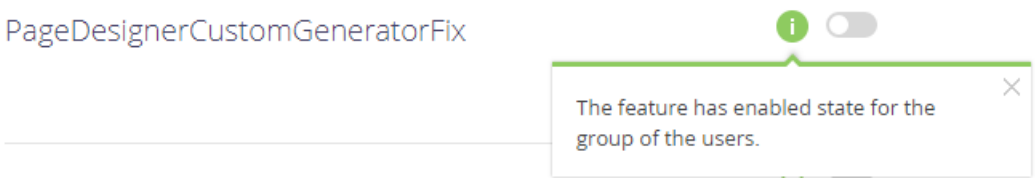
The script introduces an extra feature into the system, `PageDesignerCustomGeneratorFix`, and enables it for the "All employees" user group.

3. Examine the `generateCustomItem()` method in the `ViewModelSchemaDesignerViewGenerator` module. The method must look like this:

```
generateCustomItem: function(config) {
    if (Terrasoft.Features.getIsEnabled("PageDesignerCustomGeneratorFix")) {
        if (config) {
            delete config.generator;
        }
        return this.generateStandardItem(config);
    } else {
        return this.generateLabel({
            caption: config.name
        });
    }
}
```

Note. If the `generateCustomItem()` method is different from the one specified above, you must replace the `ViewModelSchemaDesignerViewGenerator` class and change the `generateCustomItem()` method.

4. On the newly added FeaturesPage, check the value of the `PageDesignerCustomGeneratorFix` setting. When the mouse pointer hovers on ⓘ, a notification [*The feature has enabled state for the group of the users*] must pop up.



Locking exceptions

It is possible to disable locking for some fields and details. To do this, override the `getDisableExclusionsDetailSchemaNames()` and `getDisableExclusionsColumnTags()` methods. These methods return lists of fields and details that should not be blocked by the mechanism. The implementation of methods is available below:

```
getDisableExclusionsColumnTags: function() {
 return ["SomeField"];
}
getDisableExclusionsDetailSchemaNames: function() {
 return ["SomeDetailV2"]
}
```

More complex exception logic can be implemented by overriding the `isModelItemEnabled()` method for fields and the `isDetailEnabled()` method for details. These methods are called for each field and detail. They receive the name and return the availability signal of the field or detail. The implementation of methods is available below:

```
isModelItemEnabled: function(fieldName) {
 var condition = this.get("SomeConditionAttribute");
 if (fieldName === "ExampleField" || condition) {
  return true;
 }
 return this.callParent(arguments);
}

isDetailEnabled: function(detailName) {
 if (detailName === "ExampleDetail") {
  var exampleDate = this.get("Date");
  var dateNow = new Date(this.Ext.Date.now());
  var condition = this.Ext.Date.isDate(exampleDate) && exampleDate >= dateNow;
  return condition;
 }
 return this.callParent(arguments);
}
```

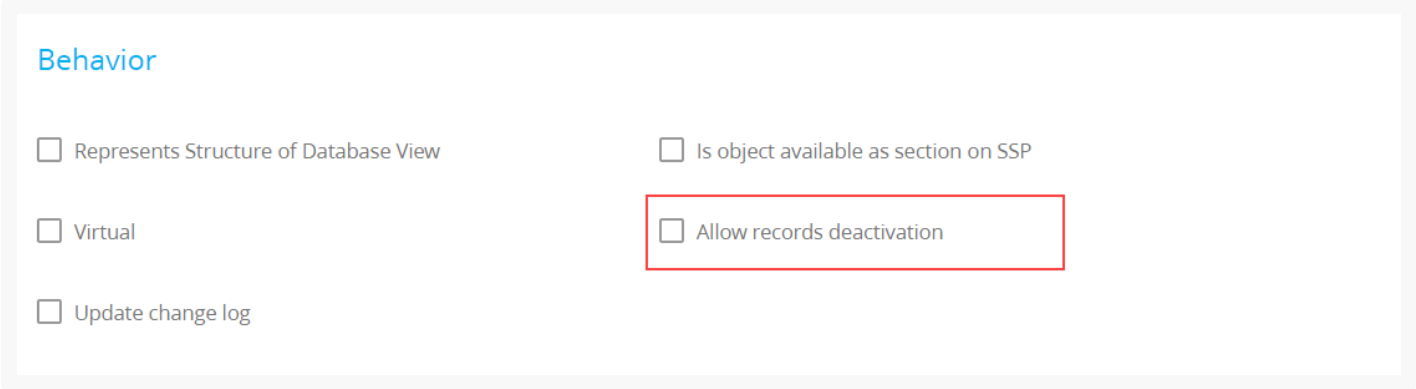
Deactivate object records

Advanced

Deactivate Creatio object records to exclude them from the business logic. For example, this can be useful if the data is outdated and no longer used.

Use the feature in the Object Designer

Use the special [*Allow record deactivation*] property in the Object Designer to enable this functionality. Creatio will apply the changes after you publish the object.



The record deactivation functionality is available for all objects, but Creatio filters the records automatically only in drop-down lists, quick filters, and on the lookup selection page. Creatio does not apply the automatic filter in advanced filters, sections, and on pages that have lookup content.

Use the feature in the code

The `UseRecordDeactivation` parameter of the `EntitySchemaQuery` class enables or disables the filters by inactive records. By default, the parameter is `false`. If you set it to `true`, the select query to the object that has record deactivation enabled will contain the filter that excludes inactive records.

Use example for client code

```
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
   rootSchemaName: "MyCustomLookup",
   useRecordDeactivation: true
});
```

Use example for server code

```
var esq = new EntitySchemaQuery(userConnection.EntitySchemaManager, "ContactType") {
    UseRecordDeactivation = true
};
esq.PrimaryQueryColumn.IsAlwaysSelect = true;
```

In this case, the resulting SQL query will look as follows:

SQL query

```
SELECT
  [ContactType].[Id] [Id]
FROM [dbo].[ContactType] [ContactType] WITH(NOLOCK)
WHERE
  [ContactType].[RecordInactive] = 0
```