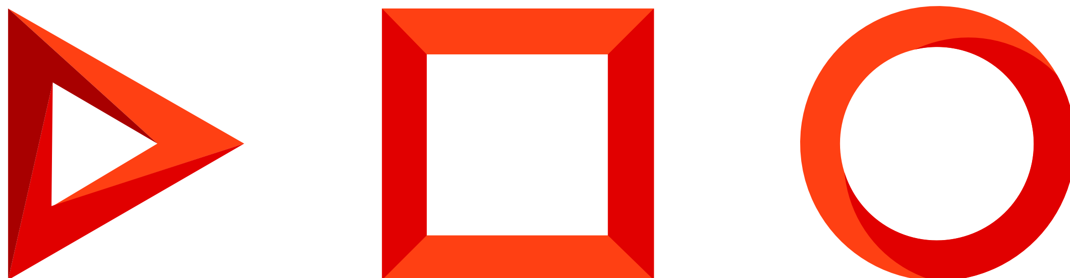
Creatio Academy

Custom additional feature

Implement a custom additional feature

Version 8.0



This documentation is provided under restrictions on use and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this documentation, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

© 2023 Creatio. All rights reserved.

Table of Contents

Implement a custom additional feature 4

1. Add a feature 4
2. Register the feature 5
3. Implement the business logic of the feature 6
4. Set up the feature status for Creatio users 9

© 2023 Creatio. All rights reserved.

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.

© 2023 Creatio. All rights reserved.

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
```

© 2023 Creatio. All rights reserved.

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. */
```

© 2023 Creatio. All rights reserved.

```
    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. */
```

© 2023 Creatio. All rights reserved.

```
        ...
    }

    /* 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.

© 2023 Creatio. All rights reserved.

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.

© 2023 Creatio. All rights reserved.