Creatio Academy

Custom UI component

Custom UI component implemented using remote module

Version 8.0

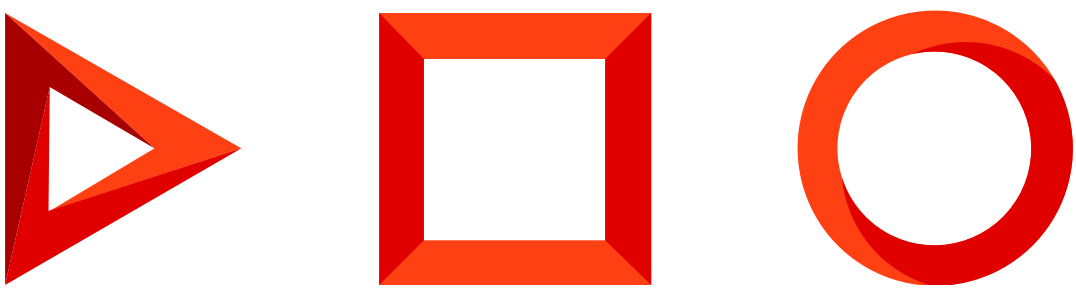


Table of Contents

Custom UI component implemented using remote module

Medium

In Creatio, you can expand the out-of-the-box set of Freedom UI page components with custom UI components.

Creatio lets you implement the following custom UI component **types**:

- Custom UI component based on a classic Creatio page element. Supported in Creatio 8.0.2 Atlas and later. Learn more in a separate article: Custom UI component based on a classic Creatio page element.
- Custom UI component implemented using remote module. Supported in Creatio 8.0.3 Atlas and later.

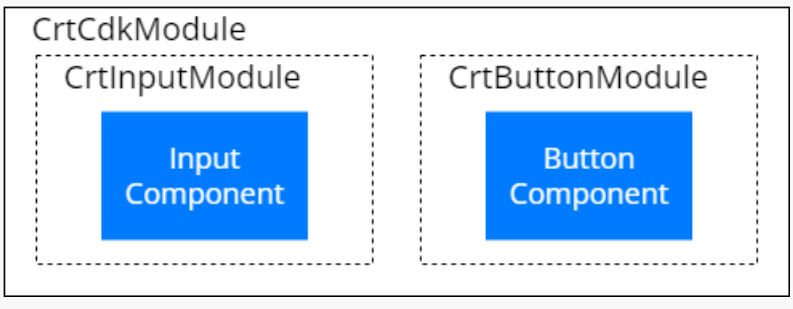
You can implement a custom UI component using remote module in Creatio version 8.0.3 Atlas and later. The feature is available for beta testing.

The development of custom UI component using remote module is based on the Module Federation plugin. The **purpose** of the Module Federation plugin is to divide the app into multiple smaller modules (components, remote modules) built independently. The plugin lets you develop custom UI components using various frameworks and library versions. Learn more in the official webpack documentation.

Note. When using the Module Federation plugin, you might experience difficulties with managing libraries that register themselves as global variables, for example, `lodash`.

The project of a remote module has a modular structure. I. e., the business logic and view elements are aggregated into modules, which, in turn, can be aggregated into higher-level modules.

View an example of the `CrtCdkModule` root project module that contains the `CrtInputModule` and `CrtButtonModule` nested modules that have their own view elements on the chart below.



To **implement a custom UI component using remote module**:

1. Create an Angular project to develop a custom UI component using remote module.
2. Create a custom UI component using remote module.

3. Implement the business logic of the custom UI component using remote module.
4. Add the custom UI component implemented using remote module to the library of the Freedom UI Designer (optional).
5. Add the custom UI component implemented using remote module to the Freedom UI page.

1. Create an Angular project to develop a custom UI component using remote module

Develop a custom UI component in a dedicated `npm` package using an external IDE. This example covers the custom UI component development in Microsoft Visual Studio Code. We recommend utilizing components created using the Angular framework. The framework requires a globally installed interface of the `@angular/cli` command line. To set up the environment for component development using Angular CLI, run the `npm install -g @angular/cli` command at the command line terminal of Microsoft Visual Studio Code.

You can create an Angular project to develop a custom UI component using remote module in the following **ways**:

- Use a *.zip archive of the Angular project that contains the template of a remote module. View a detailed example that creates an Angular project to develop a remote module in a separate article: Implement custom UI component using remote module.
- Use the Clio utility.

Create an Angular project to develop a custom UI component using remote module via a *.zip archive

1. Download and unpack the *.zip archive.

   For Creatio version **8.0.8 and later**.

   For Creatio version **8.0.3-8.0.7**.
2. Open the project in Microsoft Visual Studio Code.
3. Change the `<%projectName%>` macro of the Angular project name to the name of the package to transfer the remote module. Enter the package name in `snake case`, for example, process_designer. Change the macro in every project file.
4. Change the `<%vendorPrefix%>` prefix macro to the object name prefix, for example, `usr`. The prefix can contain up to 50 lowercase Latin characters. Change the macro in every project file.
5. Install the `npm` modules. To do this, run the `npm i` command at the command line terminal of Microsoft Visual Studio Code. The installation might take some time.
6. Install or update the `@creatio-devkit/common` library (optional).

   - Run the `npm install @creatio-devkit/common` command at the command line terminal of Microsoft Visual Studio Code to **install the release version of the library**.
   - Run the `npm update @creatio-devkit/common` command at the command line terminal of Microsoft Visual Studio Code to **update the library version**.

As a result, Microsoft Visual Studio Code will create an Angular project to develop a custom UI component using

remote module.

View a detailed example that creates an Angular project to develop a custom UI component using remote module in a separate article: Implement custom UI component using remote module.

Create an Angular project to develop a custom UI component using remote module via the Clio utility

1. Install the Clio utility (performed once). To do this, run the `dotnet tool install clio -g` command at the terminal of Microsoft Visual Studio Code.
2. Register a new Creatio instance in Clio. To do this, run the `clio reg-web-app someApplicationName -u https://mycreatio.com/ -l SomeLogin -p SomePassword` command at the Microsoft Visual Studio Code terminal.

   `someApplicationName` is the Creatio instance name.

   `https://mycreatio.com/` is the Creatio instance URL.

   `SomeLogin` is the user login to the Creatio instance.

   `SomePassword` is the user password to the Creatio instance.
3. Install the `cliogate` system package into your development environment. To do this, run the `clio install-gate someApplicationName` command at the Microsoft Visual Studio Code terminal.
4. Restart your Creatio instance. To do this, run the `clio restart someApplicationName` command at the Microsoft Visual Studio Code terminal.
5. Create a new workspace. To do this, run the `clio createw` command at the Microsoft Visual Studio Code terminal.
6. Create a project for the remote module. To do this, run the `clio ui someApplicationName -v usr --package SomePackageName --empty true` command at the Microsoft Visual Studio Code terminal. If the workspace does not contain a package that has the specified name, Microsoft Visual Studio Code creates a new package that has the specified name.

   `SomePackageName` is the name of the package to implement the remote module.
7. Install the `npm` modules. To do this, run the `npm i` command at the command line terminal of Microsoft Visual Studio Code.
8. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.
9. Upload the changes from the workspace into Creatio. To do this, run the `clio pushw -e someApplicationName` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will create an Angular project to develop a custom UI component using remote module.

2. Create a custom UI component using remote module

1. Create an Angular component in the project. To do this, run the `ng g c view-elements/some_component_name --view-encapsulation=ShadowDom` command at the command line

terminal of Microsoft Visual Studio Code.

`view-elements` specifies that the Angular component is a view element.

`some_component_name` is the custom component name.

As a result, Microsoft Visual Studio Code will add the component files to the `src/app/view-elements/some_component_name` directory.

2. Specify that the `SomeComponentNameComponent` component is a view element.

   a. Open the `some_component_name.component.ts` file.

   b. Flag the component using the `CrtViewElement` decorator.

   c. Import the functionality of the `CrtViewElement` decorator from the `@creatio-devkit/common` library to the component.

   d. Save the file.

`some_component_name.component.ts` file

```
/* Import the required functionality from the libraries. */
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
import { CrtViewElement } from '@creatio-devkit/common';

@Component({
  selector: 'usr-some_component_name',
  templateUrl: './some_component_name.component.html',
  styleUrls: ['./some_component_name.component.scss'],
  encapsulation: ViewEncapsulation.ShadowDom
})

/* Add the CrtViewElement decorator to the SomeComponentNameComponent component. */
@CrtViewElement({
  selector: 'usr-some_component_name',
  type: 'usr.SomeComponentName'
})

export class SomeComponentNameComponent implements OnInit {
  constructor() { }

  ngOnInit(): void {
  }
}
```

3. Register the `SomeComponentNameComponent` view element as a component.

   a. Open the `app.module.ts` file.

   b. Add the `SomeComponentNameComponent` view element to the `CrtModule` decorator.

   c. Register the `SomeComponentNameComponent` view element as a component, i. e., Angular Element, in the

`ngDoBootstrap()` method of the `AppModule` root module. Learn more in the Angular documentation.

d. Save the file.

`app.module.ts` file for Creatio version **8.0.8 and later**

`app.module.ts` file for Creatio version **8.0.3-8.0.7**

4. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `<%projectName%>` name specified on step 1.

View a detailed example that creates a custom UI component using remote module in a separate article: Implement custom UI component using remote module.

3. Implement the business logic of the custom UI component using remote module

1. Implement the required business logic in the component.

a. Open the `some_component_name.component.ts` file.

b. Add the `value` property to the `SomeComponentNameComponent` component class. The property manages the component value.

c. Flag the `value` property using the `Input` and `CrtInput` decorators.

d. Import the functionality of the `Input` decorator from the `@angular/core` library into the component.

e. Import the functionality of the `CrtInput` decorator from the `@creatio-devkit/common` library into the component.

f. Save the file.

`some_component_name.component.ts` file

```
/* Import the required functionality from the libraries. */
import { Component, Input, OnInit } from '@angular/core';
import { CrtInput, CrtViewElement } from '@creatio-devkit/common';
...
export class SomeComponentNameComponent implements OnInit {
  constructor() { }

  /* Add decorators to the value property. */
  @Input()
  @CrtInput()
  /* The component value. */
  public value: some_value_type;

  ngOnInit(): void {
```

```
    }
  }
```

2. Add the markup of the custom component to the `some_component_name.component.html` file.
3. Add the styles of the custom component to the `some_component_name.component.scss` file.
4. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `<%projectName%>` name specified on step 1.

Since Creatio 8.0.8 Atlas, you can create a custom converter within custom UI component implemented using remote module. Learn more in a separate article: Converters in the custom UI component implemented using remote module.

View detailed examples that implement the business logic of the custom UI component using remote module in separate articles: Implement the business logic of the custom UI component using remote module, Implement the validation in the custom UI component using remote module, Implement a custom converter in the custom UI component using remote module.

4. Add the custom UI component implemented using remote module to the library of the Freedom UI Designer (optional)

We recommend adding the custom UI component implemented using remote module to the library of the Freedom UI Designer if you are going to use the component as part of no-code development.

To **add the custom UI component implemented using remote module to the library of the Freedom UI Designer**:

1. Set up the component layout in the library of the Freedom UI Designer.

   a. Open the `some_component_name.component.ts` file.

   b. Flag the component using the `CrtInterfaceDesignerItem` decorator that has the `toolbarConfig` property. The property manages the element layout in the library of the Freedom UI Designer.

   c. Import the functionality of the `CrtInterfaceDesignerItem` decorator from the `@creatio-devkit/common` library into the component.

   d. Localize the custom UI component.

      Localize custom UI components implemented using remote module during development to save time spent on translating the finished app. The `@creatio-devkit/common` library provides the `sdk.SysValuesService` service of system variables for localization. Use the service to retrieve the current culture from the `userCulture` system variable.

      Creatio version 8.0.3 Atlas and later lets you perform the following localization **actions**:

      - Localize the metadata. To do this, follow the instructions in different section: Localize the metadata.
      - Upload the translations from static content. To do this, follow the instructions in different section: Upload the translations from static content.

   g. Upload the image to display in the library of the Freedom UI Designer and specify the path to the image in

the `icon` property. We recommend using an *.svg image.

h. Save the file.

`some_component_name.component.scss` file

```
...
/* Import the required functionality from the libraries. */
import { CrtInput, CrtInterfaceDesignerItem, CrtOutput, CrtValidationInfo, CrtValidationInput
...
/* Add the CrtViewElement decorator to the SomeComponentNameComponent component. */
@CrtInterfaceDesignerItem({
  /* Manages the element layout in the library of the Freedom UI Designer. */
  toolbarConfig: {
    /* The localizable name of the component. */
    caption: localize('SomeComponentName.Caption'),
    name: 'usr_some_component_name',
    /* The path to the component image. */
    icon: require('!!raw-loader?{esModule:false}!./some_picture_name.svg'),
    defaultPropertyValues: {
      /* The localizable name of the component in the library of the Freedom UI Designer. */
      label: 'Some component name'
    }
  }
})
...
```

2. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `<%projectName%>` name specified on step 1.

View a detailed example that adds the custom UI component implemented using remote module to the library of the Freedom UI Designer in a separate article: Add the custom UI component implemented using remote module to the library of the Freedom UI Designer.

Localize the metadata

1. Flag the metadata to localize. To do this, use the `localize()` function.

Example that uses the `localize()` function

```
@CrtInterfaceDesignerItem({
  toolbarConfig: {
    /* The localizable name of the component. */
    caption: localize('SomeComponentName.Caption'),
    name: 'usr_some_component_name',
```

```
  },
})
```

2. Specify the `localizeMetadata()` function when calling the `bootstrapCrtModule()` function. The `localizeMetadata()` function is required to translate the metadata flagged using the `localize()` function. The key to translate the value is an incoming parameter of the `localizeMetadata()` function. The function returns the translated value.

Example that uses the `localizeMetadata()` function

```
const translateService = this._injector.get(TranslateService);
bootstrapCrtModule(AppModule, {
  localizeMetadata: (key: string) => translateService.instant(key),
});
```

Upload the translations from static content

1. Find out the URL to upload the translations. To do this, use the `__webpack_public_path__` global variable. Learn more in the official webpack documentation.
2. Implement the mechanism that uploads translations.

Example that uploads the translations from static content

```
declare const __webpack_public_path__: string;

@NgModule({
  imports: [
    BrowserModule,
    HttpClientModule,
    TranslateModule.forRoot({
      defaultLanguage: 'en-US',
      loader: {
        provide: TranslateLoader,
        useFactory: (httpClient: HttpClient) => {
          return new TranslateHttpLoader(
            httpClient,
            __webpack_public_path__ + '/assets/i18n/',
            '.json'
          );
        },
        deps: HttpClient],
      },
    }),
  ],
  ...
```

5. Add the custom UI component implemented using remote module to the Freedom UI page

1. Create a custom app. To do this, follow the instruction in the user documentation: Manage apps.
2. Set up Creatio for file system development. To do this, follow the instruction in a separate article: External IDEs.
3. Download the packages to the file system.

   You can download the packages to the file system in the following **ways**:

   - using the Creatio IDE. To do this, follow the instructions in different section: Download the packages to the file system using the Creatio IDE.
   - using the Clio utility. To do this, follow the instructions in different section: Download packages to the file system using the Clio utility.
4. Open the needed page in the Freedom UI Designer.
5. Add the custom UI component to the Freedom UI page.
6. Click the </> button in the action panel of the Freedom UI Designer. After you save the page settings, Creatio opens the source code of the Freedom UI page.
7. Bind the `value` property of the custom UI component to the corresponding model attribute in the `viewConfigDiff` schema section.

   `viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "Input",
        "values": {
            ...,
            /* The property that defines the element type. */
            "type": "usr.Input",
            /* The property that defines the component value. Bound to the SomeAttributeName
            "value": "$SomeAttributeName"
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

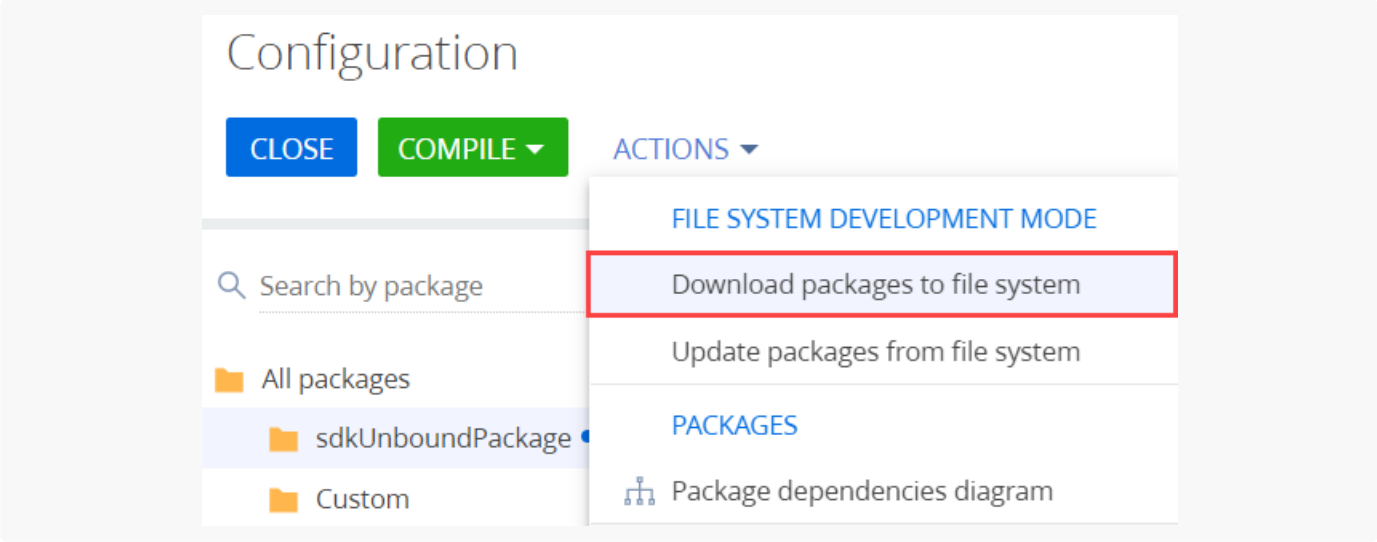
8. Click [*Save*] on the client Module Designer's toolbar.

As a result, Creatio will add the custom UI component implemented using remote module to the Freedom UI page.

View a detailed example that adds the custom UI component implemented using remote module to the Freedom UI page in a separate article: Implement custom UI component using remote module.

Download the packages to the file system using the Creatio IDE

1. Select [*Download packages to the file system*] in the [*File system development mode*] action group on the toolbar of the Creatio IDE.



2. Create a `Files/src/js` directory in the file system's app package directory.
3. Copy the build from the `dist` project directory to the `Files/src/js` directory. The path to the project build is as follows: `Files/src/js/<%projectName%>`. The `<%projectName%>` parameter must match the value specified on step 1.
4. Compile the configuration. To do this, follow the instruction in a separate article: Operations in Creatio IDE.

   As a result, Creatio will display the custom UI component in the [*Custom components*] library group of the Freedom UI Designer.

Download packages to the file system using the Clio utility

1. Copy the build from the `dist` project directory to the `Files/src/js` directory. The path to the project build is as follows: `Files/src/js/<%projectName%>`. The `<%projectName%>` parameter must match the value specified on step 1.
2. Install the package into the app. To do this, run the `clio install SomePackageName -e someApplicationName` command at the Microsoft Visual Studio Code terminal.

Implement custom UI component using remote module

Medium

The example is relevant to Creatio version 8.0.3 and later. The feature is available for beta testing.

Example. Add a custom UI component to the record page of the custom [*Requests*] section. Implement the custom UI component using a remote module created in Angular framework.

1. Create an Angular project to develop a custom UI component using remote module

Create an Angular project to develop a custom UI component using a *.zip archive of an Angular project that contains the template of a remote module.

To create an Angular project **using a *.zip archive**:

1. Download and unpack the *.zip archive.

   For Creatio version **8.0.8 and later**.

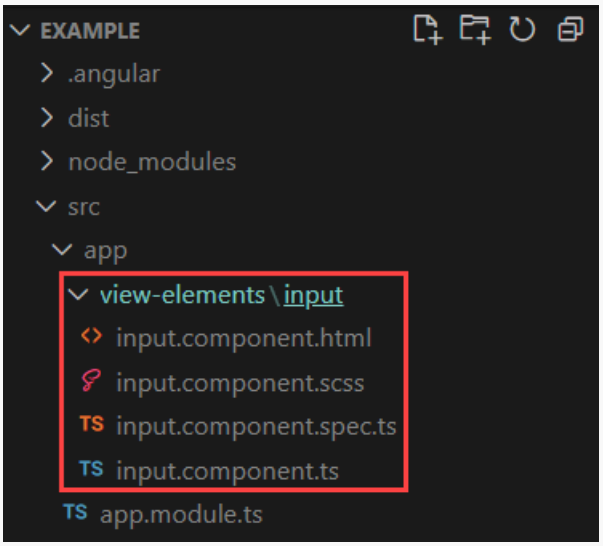
   For Creatio version **8.0.3-8.0.7**.
2. Open the project in Microsoft Visual Studio Code.
3. Change the `<%projectName%>` macro of the Angular project name to `sdk_remote_module_package` in every project file.
4. Change the `<%vendorPrefix%>` prefix macro to `usr` in every project file.
5. Install the `npm` modules. To do this, run the `npm i` command at the command line terminal of Microsoft Visual Studio Code. The installation might take some time.
6. Update the `@creatio-devkit/common` library version. To do this, run the `npm update @creatio-devkit/common` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will create an Angular project to develop a custom UI component using remote module.

2. Create a custom UI component using remote module

1. Create an Angular component in the project. To do this, run the `ng g c view-elements/input --view-encapsulation=ShadowDom` command at the command line terminal of Microsoft Visual Studio Code.

   As a result, Microsoft Visual Studio Code will add the `InputComponent` component files to the `src/app/view-elements/input` project directory.



2. Specify that the `InputComponent` component is a view element.

   a. Open the `input.component.ts` file.

   b. Flag the component using the `CrtViewElement` decorator.

   c. Import the required functionality from the libraries to the component.

   d. Save the file.

`input.component.ts` file

```
/* Import the required functionality from the libraries. */
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
import { CrtViewElement } from '@creatio-devkit/common';

@Component({
  selector: 'usr-input',
  templateUrl: './input.component.html',
  styleUrls: ['./input.component.scss'],
  encapsulation: ViewEncapsulation.ShadowDom
})

/* Add the CrtViewElement decorator to the InputComponent component. */
@CrtViewElement({
  selector: 'usr-input',
  type: 'usr.Input'
})

export class InputComponent implements OnInit {
  constructor() { }

  ngOnInit(): void {
  }
}
```

3. Register the `InputComponent` view element as a component.

   a. Open the `app.module.ts` file.

   b. Add the `InputComponent` view element to the `CrtModule` decorator.

   c. Register the `InputComponent` view element as a component, i. e., Angular Element, in the `ngDoBootstrap()` method of the `AppModule` root module.

   d. Save the file.

   `app.module.ts` file for Creatio version **8.0.8 and later**

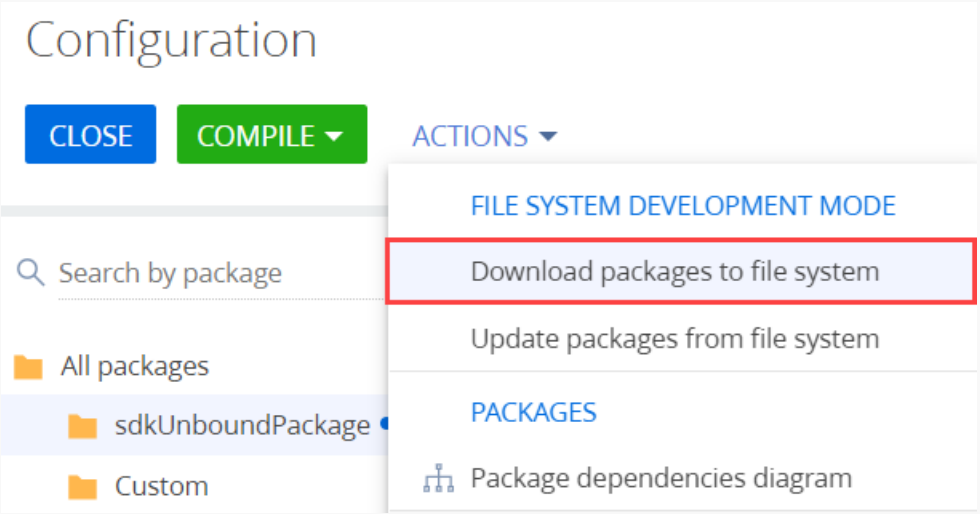
   `app.module.ts` file for Creatio version **8.0.3-8.0.7**

4. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `sdk_remote_module_package` name.

3. Add the custom UI component implemented using remote module to the Freedom UI page

1. Use the [*Records & business processes*] template to create a custom `Requests` app. To do this, follow the instruction in the user documentation: Manage apps.
2. Set up Creatio for file system development. To do this, follow the instruction in a separate article: External IDEs.
3. Download the packages to the file system using the Creatio IDE. On the Creatio IDE toolbar, select [*Download packages to the file system*] in the [*File system development mode*] action group.



4. Create a `Files/src/js` directory in the file system's `UsrRequests` package directory.

5. Copy the build from the `dist` project directory to the `Files/src/js` directory. The path to the project build is as follows: `Files/src/js/sdk_remote_module_package`.
6. Compile the configuration. To do this, follow the instruction in a separate article: Operations in Creatio IDE.
7. Open the `Requests` app page workspace and go to the [*Requests form page*] page.
8. Click the  button in the action panel of the Freedom UI Designer. After you save the page settings, Creatio opens the source code of the Freedom UI page.
9. Add the configuration object of the custom component to the `viewConfigDiff` schema section.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "UsrInput",
        "values": {
            "layoutConfig": {
                "column": 1,
                "row": 2,
                "colSpan": 1,
                "rowSpan": 1
            },
            /* The property that defines the element type. */
            "type": "usr.Input",
        },
        "parentName": "SideAreaProfileContainer",
        "propertyName": "items",
        "index": 1
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

Complete source code of the page schema

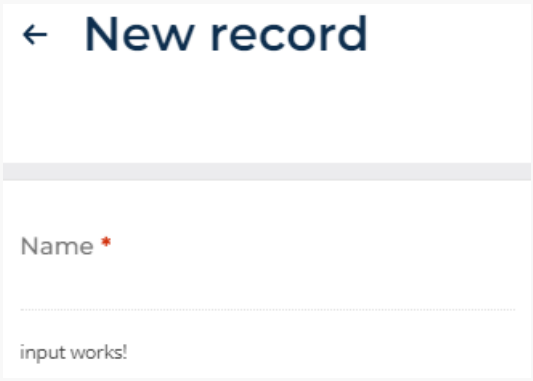
10. Click [*Save*] on the client Module Designer's toolbar.

Outcome of the example

To **view the outcome of the example**:

1. Open the `Requests` app page and click [*Run app*].
2. Click [*New*] on the `Requests` app toolbar.

As a result, Creatio will display the custom UI component on the request page. The custom UI component is implemented using a remote module created in Angular framework.



You can proceed to implement the business logic. For example, you can implement validation or a custom converter. Learn more in separate articles: Implement the business logic of the custom UI component using remote module, Implement the validation in the custom UI component using remote module, Implement a custom converter in the custom UI component using remote module.

Implement the business logic of the custom UI component using remote module

Medium

The example is relevant to Creatio version 8.0.3 and later.

Example. Add a custom UI component to the record page of the custom [*Requests*] section. The custom UI component must be an input. The name and value of the custom input must match the [*Name*] field. Implement the custom UI component using a remote module created in Angular framework.

1. Implement a custom UI component using remote module

To **implement a custom UI component using remote module**, follow the instruction in a separate article: Implement custom UI component using remote module.

2. Implement an input within remote module

1. Implement an input in the component.
    a. Open the `input.component.ts` file.
    b. Add the `value` property to the `InputComponent` component class. The property manages the input value.
    c. Flag the `value` property using the `Input` and `CrtInput` decorators.
    d. Add the `label` property to the `InputComponent` component class. The property manages the input name.
    e. Flag the `label` property using the `Input` and `CrtInput` decorators.

f. Import the required functionality from the libraries into the component.

g. Save the file.

`input.component.ts` file

```
/* Import the required functionality from the libraries. */
import { Component, Input, OnInit } from '@angular/core';
import { CrtInput, CrtViewElement } from '@creatio-devkit/common';
...
export class InputComponent implements OnInit {
  constructor() { }

  /* Add decorators to the value property. */
  @Input()
  @CrtInput()
  /* The input value. */
  public value: string = '';

  /* Add decorators to the label property. */
  @Input()
  @CrtInput()
  /* The input name. */
  public label!: string;

  ngOnInit(): void {
  }
}
```

2. Track changes of input values.

a. Open the `input.component.ts` file.

b. Add the `EventEmitter<string>()` event to the `InputComponent` component class. The event tracks input value changes.

c. Flag the `EventEmitter<string>()` using the `Output` and `CrtOutput` decorators.

d. Import the required functionality from the libraries into the component.

e. Save the file.

`input.component.ts` file

```
/* Import the required functionality from the libraries. */
import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
import { CrtInput, CrtOutput, CrtViewElement } from '@creatio-devkit/common';
...
export class InputComponent implements OnInit {
  ...
```

```
  /* Add decorators to the EventEmitter<string>() event. */
  @Output()
  @CrtOutput()
  /* Track input value changes. */
  public valueChange = new EventEmitter<string>();
  ...
}
```

Complete source code of the `input.component.ts` file

3. Add the markup of the custom component.
   a. Open the `input.component.html` file.
   b. Add the markup.
   c. Save the file.

`input.component.html` file

```
<div class="wrapper">
    <label class="label">{{label}}</label>
    <input
        #input
        class="input"
        type="text"
        [value]="value"
        (keyup)="valueChange.emit(input.value)"
    />
</div>
```

4. Add the styles of the custom component.
   a. Open the `input.component.scss` file.
   b. Add the styles.
   c. Save the file.

`input.component.scss` file

```
.wrapper {
    display: flex;
    flex-direction: row;
    gap: 10px;
    padding: 10px;
    align-items: center;
}
```

5. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `sdk_remote_module_package` name.

3. Add the input to the Freedom UI page

1. Add the custom UI component implemented using remote module to the Freedom UI page. To do this, follow the instruction in a separate article: Implement custom UI component using remote module.
2. Add the input implemented in remote module to the `viewConfigDiff` schema section.
   - Bind the `label` property of the `UsrInput` element to the localizable string of the [*Name*] field name.
   - Bind the `value` property of the `UsrInput` element to the `$UsrName` model attribute.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "UsrInput",
        "values": {
            ...,
            /* The property that defines the input name. Bound to the UsrName attribute. */
            "label": "$Resources.Strings.UsrName",
            /* The property that defines the input value. Bound to the UsrName attribute. */
            "value": "$UsrName",
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

Complete source code of the page schema

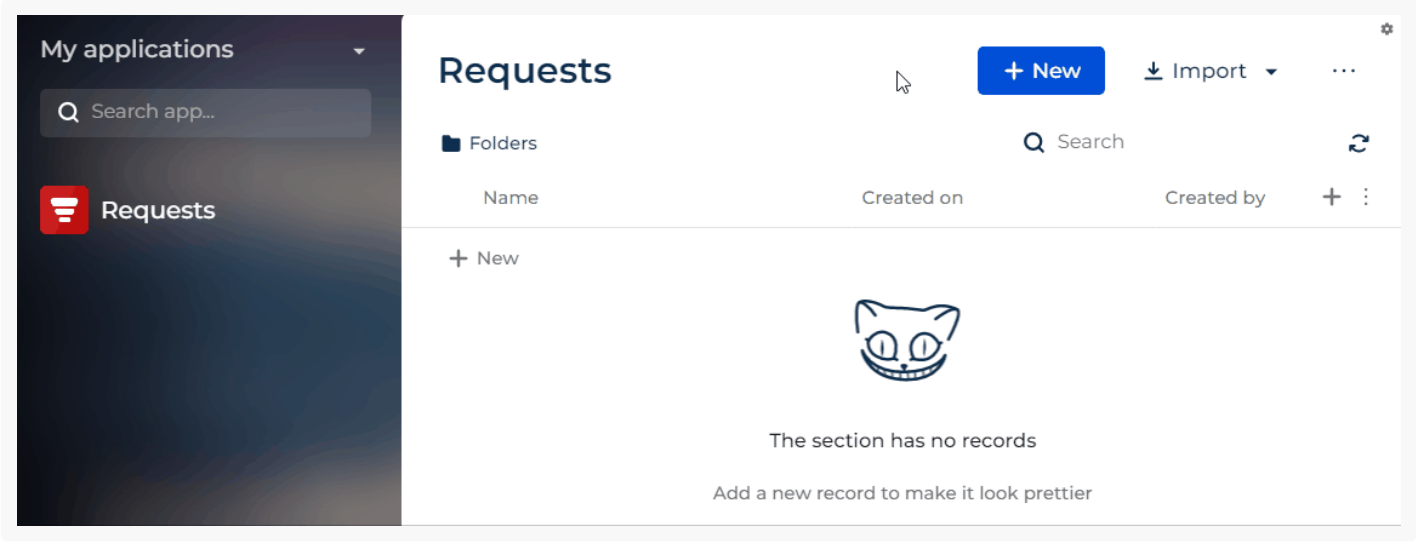
3. Click [*Save*] on the client Module Designer's toolbar.

Outcome of the example

To **view the outcome of the example**:

1. Open the `Requests` app page and click [*Run app*].
2. Click [*New*] on the `Requests` app toolbar.
3. Enter "`Test`" in the [*Name*] input.

As a result, Creatio will display the custom input on the request page. The name and value of the custom input match the [*Name*] field. The input is implemented using a remote module created in Angular framework.



You can set up input validation if needed. Learn more in a separate article: Implement the validation in the custom UI component using remote module.

Implement the validation in the custom UI component using remote module

Medium

The example is relevant to Creatio version 8.0.3 and later.

Example. Add a validator that checks whether the custom UI component is filled out to the record page of the custom [*Requests*] section. The custom UI component must be an input. Implement the custom UI component using a remote module created in Angular framework.

1. Implement a custom UI component using remote module

To **implement a custom UI component using remote module**, follow the instruction in a separate article: Implement custom UI component using remote module.

2. Implement an input within remote module

To **implement an input within remote module**, follow the instruction in a separate article: Implement the business logic of the custom UI component using remote module.

3. Implement the input validation

1. Add the validity flag of the bound attribute to the component.

   a. Open the `input.component.ts` file.

   b. Add the `valueValidationInfo` property to the `InputComponent` component class. The property displays information that the bound attribute is invalid.

   c. Flag the `valueValidationInfo` property using the `Input` and `CrtValidationInput` decorators.

   d. Import the required functionality from the libraries into the component.

   e. Save the file.

   `input.component.ts` file

```
/* Import the required functionality from the libraries. */
import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
import { CrtInput, CrtOutput, CrtValidationInfo, CrtValidationInput, CrtViewElement } from '@
...
export class InputComponent implements OnInit {
  ...
  /* Add the decorators to the valueValidationInfo property. */
  @Input()
  @CrtValidationInput()
  /* Display information that the input value is invalid. */
  public valueValidationInfo!: CrtValidationInfo;
  ...
}
```

   Complete source code of the `input.component.ts` file

2. Add the markup of the custom component if the input value is invalid.

   a. Open the `input.component.html` file.

   b. Add the markup.

   c. Save the file.

   `input.component.html` file

```
<div class="wrapper">
    <label class="label">{{label}}</label>
    <input
        ...
        [class.invalid]="
        valueValidationInfo &&
        !valueValidationInfo.valid && valueValidationInfo.touched
        "
```

```
        ...
    />
</div>
```

Complete source code of the `input.component.html` file

3. Add the styles of the custom component.

   a. Open the `input.component.scss` file.

   b. Add the markup.

   c. Save the file.

`input.component.scss` file

```
.wrapper {
    ...
    .input.invalid {
        background-color: #FDD8CF;
    }
}
```

Complete source code of the `input.component.scss` file

4. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

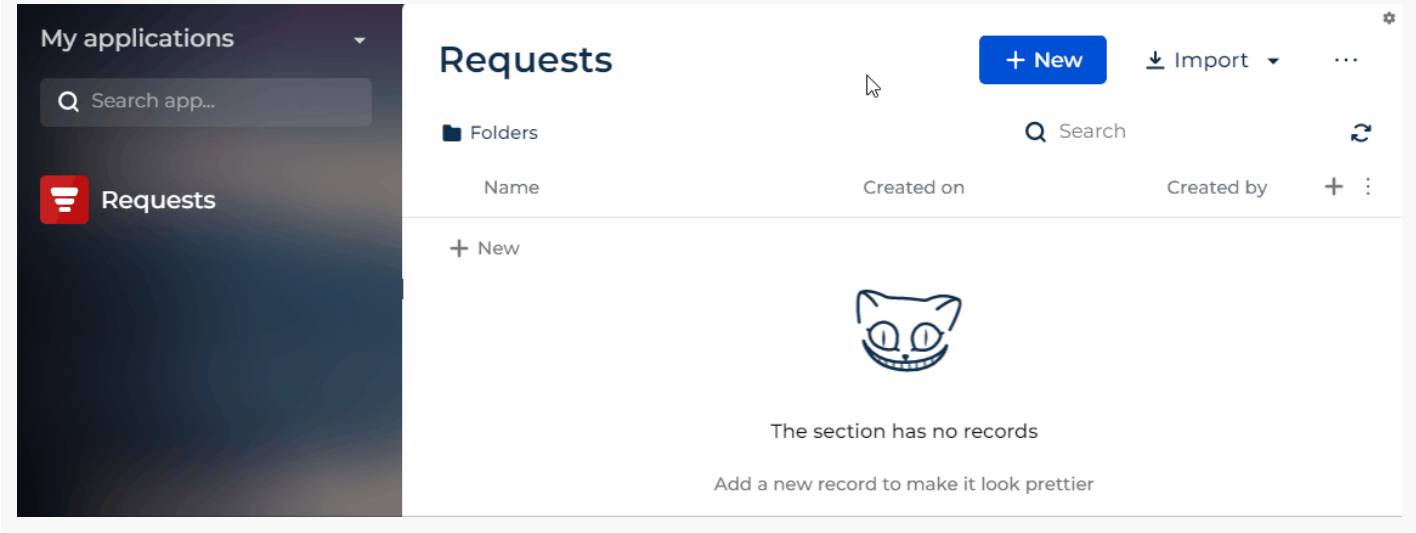
As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `sdk_remote_module_package` name.

Outcome of the example

To **view the outcome of the example**:

1. Open the `Requests` app page and click [*Run app*].
2. Click [*New*] on the `Requests` app toolbar.
3. Enter "`Test`" in the [*Name*] input.
4. Clear the custom input.

As a result, Creatio will display the custom input on the request page. The name and value of the custom input match the [*Name*] field. The input is implemented using a remote module created in Angular framework.



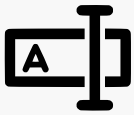
You can proceed to add the custom UI component implemented using remote module to the Freedom UI page. Learn more in a separate article: Add the custom UI component implemented using remote module to the library of the Freedom UI Designer.

Add the custom UI component implemented using remote module to the library of the Freedom UI Designer

Medium

The example is relevant to Creatio version 8.0.3 and later.

Example. Add the custom UI component to the library of the Freedom UI Designer. Implement the custom component using a remote module created in Angular framework. Use the image provided below.



1. Implement a custom UI component using remote module

To **implement a custom UI component using remote module**, follow the instruction in a separate article: Implement custom UI component using remote module.

2. Implement an input within remote module

To **implement an input within remote module**, follow the instruction in a separate article: Implement the business logic of the custom UI component using remote module.

3. Implement the input validation

To **implement the input validation**, follow the instruction in a separate article: Implement the validation in the custom UI component using remote module.

4. Add the custom UI component to the library of the Freedom UI Designer

1. Set up the component layout in the library of the Freedom UI Designer.
   a. Open the `input.component.ts` file.
   b. Flag the component using the `CrtInterfaceDesignerItem` decorator that has the `toolbarConfig` property. The property manages the element layout in the library of the Freedom UI Designer.
   c. Import the required functionality from the libraries into the component.
   d. Upload the image to display in the library of the Freedom UI Designer and specify the path to the image in the `icon` property. In this example, the image is in the `input` directory.
   e. Transfer the icon in the `string` format. To do this, run the `npm install raw-loader --save-dev` command at the command line terminal of Microsoft Visual Studio Code.
   f. Save the file.

`input.component.ts` file

```
...
/* Import the required functionality from the libraries. */
import { CrtInput, CrtInterfaceDesignerItem, CrtOutput, CrtValidationInfo, CrtValidationInput
...
/* Add the CrtViewElement decorator to the InputComponent component. */
@CrtInterfaceDesignerItem({
  /* Manage the element layout in the library of the Freedom UI Designer. */
  toolbarConfig: {
    caption: 'Custom Input',
    name: 'CustomInput',
    /* The path to the component image. */
    icon: require('!!raw-loader?{esModule:false}!./icon.svg'),
    defaultPropertyValues: {
      label: 'Custom input'
    }
  }
})
...
```

Complete source code of the `input.component.ts` file

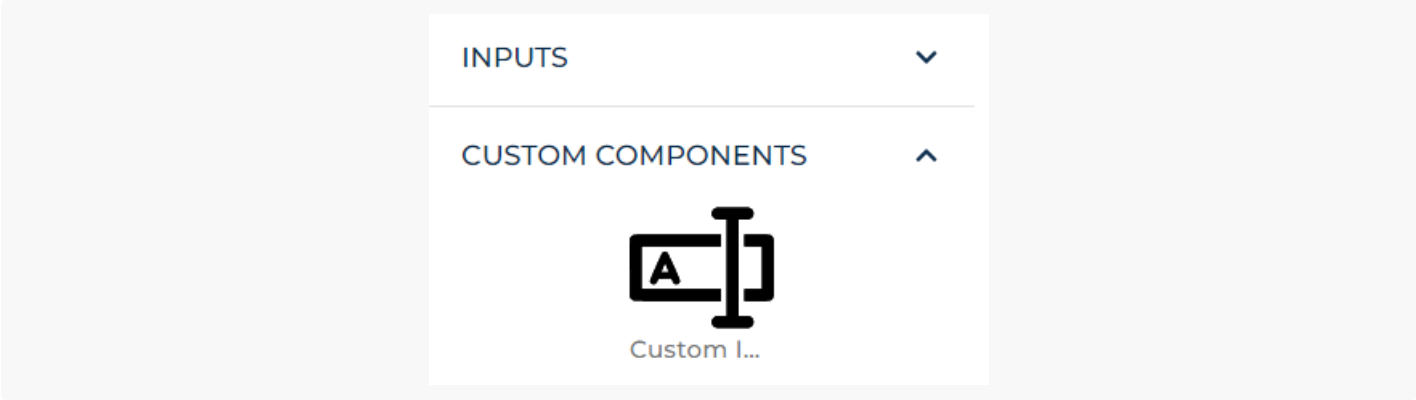
2. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `sdk_remote_module_package` name.

Outcome of the example

To **view the outcome of the example**, open the [*Requests form page*] page in the working area of the `Requests` app page.

As a result, Creatio will display the [*Custom input*] component in the [*Custom components*] library group of the Freedom UI Designer.



Creatio will display the custom UI component when you add it to the canvas of the Freedom UI Designer as well.