Creatio Academy

Custom converter

Custom converter implemented using remote module

Version 8.0

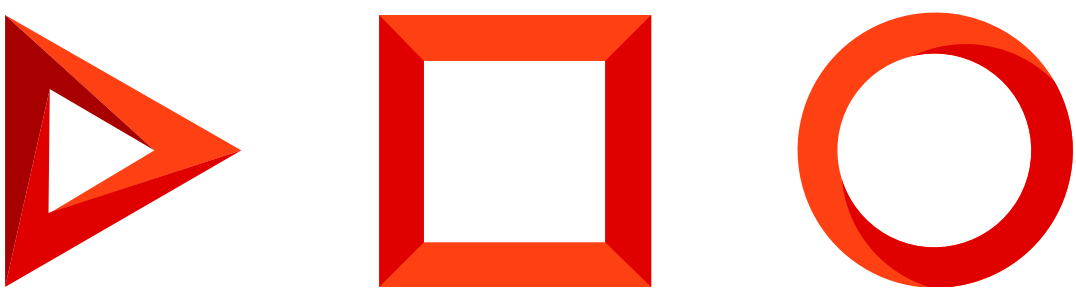


Table of Contents

Custom converter implemented using remote module

Medium

You can implement a custom converter within remote module in Creatio version 8.0.8 Atlas and later.

To **implement a custom converter using remote module**:

1. Create an Angular project to develop a custom converter using remote module.
2. Create a custom converter using remote module.
3. Implement the business logic of the custom converter using remote module.
4. Add the converter implemented using remote module to the Freedom UI page.

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

Develop a custom converter in a dedicated `npm` package using an external IDE. This example covers the custom converter development in Microsoft Visual Studio Code.

You can create an Angular project to develop a custom converter in multiple ways, similarly to creating an Angular project to develop a custom UI component using remote module. To do this, follow the instruction in a separate article: Custom UI component implemented using remote module.

2. Create a custom converter using remote module

You can implement synchronous or asynchronous custom converters. The procedure for implementing a custom converter is similar for various types of converters.

1. Create an Angular class in the project. To do this, run the `ng g class some_converter_name.converter` command at the command line terminal of Microsoft Visual Studio Code.

   `some_converter_name.converter` is an arbitrary class name.

   As a result, Microsoft Visual Studio Code will add the `SomeConverterNameConverter` class files to the `src/app/` project directory.

2. Specify that the `SomeConverterNameConverter` class is a converter.

   a. Open the `some_converter_name.converter.ts` file.

   b. Create the `SomeConverterNameConverter` class that implements the interface from the `@creatio-devkit/common` library. The `SomeConverterNameConverter` class implements one of the following interfaces:

- `Converter<V, R>` **for synchronous converters**
- `Converter<V, Promise<R>>` **for asynchronous converters**

`V` is type of convertible attribute.

`R` is type of conversion result.

e. Flag the `SomeConverterNameConverter` class using the `CrtConverter` decorator.

f. Import the required functionality from the libraries to the class.

g. Save the file.

`some_converter_name.converter.ts` file

```
/* Import the required functionality from the libraries. */
import { Converter, CrtConverter } from "@creatio-devkit/common";

/* Add the CrtConverter decorator to the SomeConverterNameConverter interface. */
@CrtConverter({
  type: 'usr.SomeConverterNameConverter',
})

/* The class for synchronous converters. */
export class SomeConverterNameConverter implements Converter<string, string> {
}

/* The class for asynchronous converters. */
export class SomeConverterNameConverter implements Converter<string, Promise<string>> {
}
```

3. Inject dependencies.

a. Open the `some_converter_name.converter.ts` file.

b. Create the `SOME_TOKEN_NAME` instance of the `InjectionToken` class (optional). Use the token to work with Angular DI (dependency injection). You can use a client web service instead of the token.

c. Flag the `SOME_TOKEN_NAME` token using the `CrtInject` decorator.

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

e. Save the file.

`some_converter_name.converter.ts` file

```
/* Import the required functionality from the libraries. */
import { InjectionToken } from "@angular/core";
import { Converter, CrtInject } from "@creatio-devkit/common";

/* The SOME_TOKEN_NAME token that works with Angular DI (dependency injection). */
export const SOME_TOKEN_NAME = new InjectionToken<string>('SOME_TOKEN_NAME');
```

```
...
/* The class for synchronous converters. */
export class SomeConverterNameConverter implements Converter<string, string> {
  constructor(@CrtInject(SOME_TOKEN_NAME) private _separator: string) { };
}

/* The class for asynchronous converters. */
export class SomeConverterNameConverter implements Converter<string, Promise<string>> {
  constructor(@CrtInject(SOME_TOKEN_NAME) private _separator: string) { };
}
```

4. Register the `SomeConverterNameConverter` converter as a converter and the `SOME_TOKEN_NAME` token.

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

   b. Add the `SomeConverterNameConverter` converter to the `converters` section in the `CrtModule` decorator.

   c. Add the configuration object to the `providers` section in the `NgModule` decorator.

   - Set the `provide` property to `SOME_TOKEN_NAME`.
   - Set the `useValue` property to space.

   f. Implement the `resolveDependency()` method in the `bootstrapCrtModule()` method of the `AppModule` root module. The `bootstrapCrtModule()` method registers the `SomeConverterNameConverter` converter flagged using the `CrtModule` decorator. The `resolveDependency()` method receives the dependencies of the `SomeConverterNameConverter` converter.

   g. Import the required functionality from the libraries into the class.

   h. Save the file.

`app.module.ts` file

```
/* Import the required functionality from the libraries. */
import { DoBootstrap, Injector, NgModule, ProviderToken } from '@angular/core';
import { bootstrapCrtModule, CrtModule } from '@creatio-devkit/common';
import { SomeConverterNameConverter, SOME_TOKEN_NAME } from './some_converter_name.converter'

@CrtModule({
  ...,
  /* Specify that SomeConverterNameConverter is a converter. */
  converters: [SomeConverterNameConverter]
})
@NgModule({
  ...,
  providers: [
    {
      provide: SOME_TOKEN_NAME,
      useValue: ' '
```

```
    },
  ],
})
export class AppModule implements DoBootstrap {
  constructor(private _injector: Injector) {}

  ngDoBootstrap(): void {

    /* Bootstrap CrtModule definitions. */
    bootstrapCrtModule('some_package_name', AppModule, {
      resolveDependency: (token) => this._injector.get(<ProviderToken<unknown>>token),
    });
  }
}
```

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 `<%projectName%>` name specified on step 1.

3. Implement the business logic of the custom converter using remote module

1. Implement the conversion of an incoming value.
   a. Open the `some_converter_name.converter.ts` file.
   b. Implement a method of the `SomeConverterNameConverter` interface. Implement one of the following methods:
      - `convert(value: V, context: ViewModelContext, ...params: unknown[]): R;` **for synchronous converters**
      - `async convert(value: V, context: ViewModelContext, ...params: unknown[]): R;` **for asynchronous converters**

      `V` is the type of convertible attribute.

      `R` is the type of conversion result.
   e. Import the required functionality from the libraries into the class.
   f. Save the file.

`some_converter_name.converter.ts` file

```
/* Import the required functionality from the libraries. */
import { Converter, ViewModelContext } from "@creatio-devkit/common";

...
/* The class for synchronous converters. */
```

```
export class SomeConverterNameConverter implements Converter<string, string> {

  /* Implement the convert() method for synchronous converters. */
  public convert(value: string, context: ViewModelContext, ...params: string[]): string {
    return [value, ...params].join(this._separator);
  }
}

/* The class for asynchronous converters. */
export class SomeConverterNameConverter implements Converter<string, Promise<string>> {

  /* Implement the convert() method for asynchronous converters. */
  public async convert(value: string, context: ViewModelContext, ...params: string[]): Promis
    /* Implement some async action. */
    ...;

    await new Promise(function(resolve) {
      setTimeout(resolve, 100);
    });
    return [value, ...params].join(this._separator);
  }
}
```

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.

4. Add the converter implemented using remote module to the Freedom UI page

1. Repeat steps 1-6 of the procedure to implement a custom UI component using remote module.
2. Bind the custom converter implemented using a remote module to the `caption` property of the corresponding model attribute in the `viewConfigDiff` schema section. Creatio lets you set converter parameters. Learn more in a separate article: Customize page fields.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "SomeComponentName",
        "values": {
            ...,
```

```
            /* Apply converter to the $SomeAttributeName attribute. Use a synchronous or asyn
            "caption": "$SomeAttributeName | usr.SomeConverterNameConverter : 'SomeConverterP
            ...
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

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

As a result, Creatio will apply the converter to the element on the Freedom UI page.

Implement a custom converter using remote module

 Medium

The example is relevant to Creatio version 8.0.8 and later.

Example. Add a converter that adds the `is the request name` string to the [*Name*] field value of the custom [*Requests*] section. Display the converted value in the [*Label*] type component. Implement the synchronous converter using a remote module created in Angular framework.

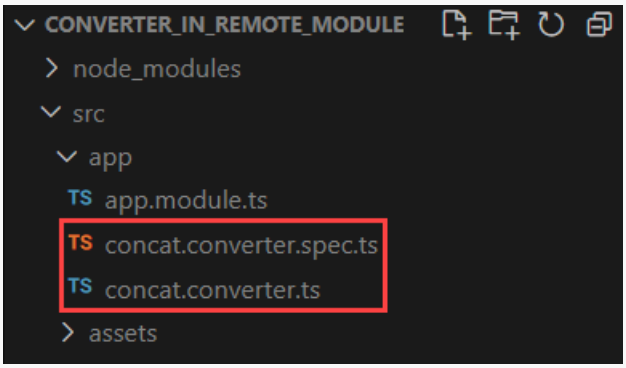
1. Create an Angular project to develop a custom converter using remote module

To **create an Angular project to develop a custom converter using remote module**, follow the instruction in a separate article: Implement custom UI component using remote module.

2. Create a custom converter using remote module

1. Create an Angular class in the project. To do this, run the `ng g class concat.converter` command at the command line terminal of Microsoft Visual Studio Code.

   As a result, Microsoft Visual Studio Code will add the `ConcatConverter` class files to the `src/app/` project directory.



2. Specify that the `ConcatConverter` class is a converter.

   a. Open the `concat.converter.ts` file.

   b. Create the `ConcatConverter` class that implements the `Converter<string, string>` interface from the `@creatio-devkit/common` library.

   c. Flag the `ConcatConverter` class using the `CrtConverter` decorator.

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

   e. Save the file.

`concat.converter.ts` file

```
/* Import the required functionality from the libraries. */
import { Converter, CrtConverter } from "@creatio-devkit/common";

/* Add the CrtConverter decorator to the ConcatConverter interface. */
@CrtConverter({
  type: 'usr.ConcatConverter',
})

export class ConcatConverter implements Converter<string, string> {
}
```

3. Inject dependencies.

   a. Open the `concat.converter.ts` file.

   b. Create the `CONCAT_SEPARATOR_TOKEN` instance of the `InjectionToken` class. The token works with Angular DI (dependency injection).

   c. Flag the `CONCAT_SEPARATOR_TOKEN` token using the `CrtInject` decorator.

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

   e. Save the file.

`concat.converter.ts` file

```
/* Import the required functionality from the libraries. */
```

```
import { InjectionToken } from "@angular/core";
import { Converter, CrtInject } from "@creatio-devkit/common";

/* The CONCAT_SEPARATOR_TOKEN token that works with Angular DI (dependency injection). */
export const CONCAT_SEPARATOR_TOKEN = new InjectionToken<string>('CONCAT_SEPARATOR_TOKEN');

...
export class ConcatConverter implements Converter<string, string> {
  constructor(@CrtInject(CONCAT_SEPARATOR_TOKEN) private _separator: string) { };
}
```

4. Register the `ConcatConverter` converter as a converter and the `CONCAT_SEPARATOR_TOKEN` token.
   a. Open the `app.module.ts` file.
   b. Add the `ConcatConverter` converter to the `converters` section in the `CrtModule` decorator.
   c. Add the configuration object to the `providers` section in the `NgModule` decorator.
      - Set `provide` property to `CONCAT_SEPARATOR_TOKEN`.
      - Set `useValue` property to space.
   f. Implement the `resolveDependency()` method in the `bootstrapCrtModule()` method of the `AppModule` root module. The `bootstrapCrtModule()` method registers the `ConcatConverter` converter flagged using the `CrtModule` decorator. The `resolveDependency()` method receives the dependencies of the `ConcatConverter` converter.
   g. Import the required functionality from the libraries into the class.
   h. Save the file.

`app.module.ts` file

```
/* Import the required functionality from the libraries. */
import { DoBootstrap, Injector, NgModule, ProviderToken } from '@angular/core';
import { bootstrapCrtModule, CrtModule } from '@creatio-devkit/common';
import { ConcatConverter, CONCAT_SEPARATOR_TOKEN } from './concat.converter';

@CrtModule({
  ...,
  /* Specify that ConcatConverter is a converter. */
  converters: [ConcatConverter]
})
@NgModule({
  ...,
  providers: [
    {
      provide: CONCAT_SEPARATOR_TOKEN,
      useValue: ' '
    },
```

```
  ],
})
export class AppModule implements DoBootstrap {
  constructor(private _injector: Injector) {}

  ngDoBootstrap(): void {

    /* Bootstrap CrtModule definitions. */
    bootstrapCrtModule('sdk_remote_module_package', AppModule, {
      resolveDependency: (token) => this._injector.get(<ProviderToken<unknown>>token),
    });
  }
}
```

Complete source code of the `app.module.ts` file

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. Implement the business logic of the custom converter using remote module

1. Implement the conversion of an incoming value.
   a. Open the `concat.converter.ts` file.
   b. Implement the `convert(value: string, context: ViewModelContext, ...params: string[]): string;` method of the `ConcatConverter` interface.
   c. Import the required functionality from the libraries into the class.
   d. Save the file.

`concat.converter.ts` file

```
/* Import the required functionality from the libraries. */
import { Converter, ViewModelContext } from "@creatio-devkit/common";

...
export class ConcatConverter implements Converter<string, string> {
  ...

  /* Implement the convert() method. */
  public convert(value: string, context: ViewModelContext, ...params: string[]): string {
    return [value, ...params].join(this._separator);
```

```
  }
}
```

Complete source code of the `concat.converter.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.

4. Add the converter implemented using remote module to the Freedom UI page

1. Repeat steps 1-7 of the procedure to Implement custom UI component using remote module.
2. Add a [*Label*] type component to the working area of the Freedom UI Designer.
3. 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.
4. Bind the `caption` property of the `Label` element to the `$UsrName` model attribute in the `viewConfigDiff` schema section. `$UsrName` is the value of the [*Name*] field. Add the `usr.Converter` converter with the `'is the request name'` parameter to the `$UsrName` attribute.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "Label",
        "values": {
            ...,
            /* Apply the usr.ConcatConverter converter to the $UsrName attribute. */
            "caption": "$UsrName | usr.ConcatConverter : 'is the request name'",
            ...
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

Complete source code of the page schema

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

As a result, Creatio will add the converter to the Freedom UI page.

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 "`REQ-001`" in the [*Name*] input.

As a result, Creatio will display the converted value on the request page. The synchronous converter is implemented using a remote module created in Angular framework.