Creatio Academy

Page customization

Version 8.0

Table of Contents

Freedom UI page customization basics

Beginner

Configure the business logic of Freedom UI pages in the `validators`, `converters`, and `handlers` sections of the client schemas. We recommend using no-code tools to set up business logic. Learn more in the user documentation: Freedom UI Designer. If no-code tools are not sufficient to implement your customization, we recommend setting up business logic in the source code of the Freedom UI page schema. Learn more about creating a Freedom UI page in a separate article: Client module.

For example, you can use a Freedom UI page to implement the following **business logic**:

- element visibility
- element locking
- element requirement
- element filtering
- field population
- Creatio data querying
- HTTP request sending
- page navigation

Learn more about pages in a separate article: Freedom UI page.

Page customization procedure

1. Set up an **app page**.

   a. Create a custom app. To do this, follow the procedure in the user documentation: Create a custom app.

   b. Add one or more elements whose business logic to set up. To do this, follow the procedure in the user documentation: Set up the app UI.

2. Set up the **business logic of a page item**.

   Configure the business logic in the client schema of the Freedom UI page. To **open the client module schema**, click the </> button on the action panel of the Freedom UI Designer on the corresponding page.

   The source code of the Freedom UI page opens after you save the page settings. Perform the setup in the corresponding client schema sections of the Freedom UI page Learn more in a separate article: Client schema.

Query handlers are the preferred way to customize the page. **Query handlers** are items of the `HandlerChain` mechanism that lets you describe business logic as an action request and chain of handlers. You can manage data sources using the `handlers` schema section. Query examples: page readiness, data load and save, business process launch.

Creatio 8 Atlas lets you organize query handlers in event chains. For example, trigger the base Creatio save handler first and execute the custom page logic later upon the page saver query.

To **limit the scope of a query handler**, fill out the `scopes` property of the client module with the names of the client schemas for which the handler must work.

View detailed examples of handler invocation in a separate block: Examples.

Close the WebSocket when destroying the `View` of the model

To **close the WebSocket when destroying** the `View` of the **model**, add a custom implementation of the `crt.HandleViewModelDestroyRequest` system query handler to the `handlers` schema section. The handler is executed when the `View` of the model is destroyed (for example, when you open another page). Designed to destroy resources. We do not recommend writing asynchronous code in the handler (server calls, timeouts, etc.) except for reading the value of attributes.

View an example of the `crt.HandleViewModelDestroyRequest` query handler that closes the custom `SomeWebSocket` WebSocket below.

`handlers` schema section

Manage the system setting values on a page

Medium

Example. Display the following on the record page of the [*Requests*] custom section:

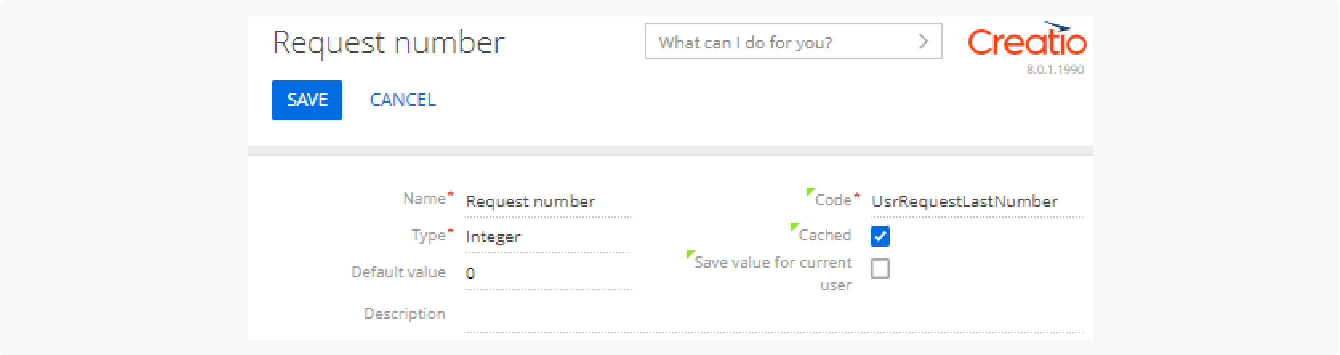
- User’s city.
- The number of the last created record in the [*Requests*] section, increased by 1. Increase the number when creating a new request or copying an existing one.

Use the values of custom system settings.

1. Set up page UI

1. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the procedure in the user documentation: Create a custom app.
2. Add a **system setting that stores the request number**.
    a. Click [*Run app*] on the `Requests` app page.
    b. Click ⚙ to open the System Designer. Click [*System settings*] in the [*System setup*] block.
    c. Click [*Add setting*] on the section toolbar.
    d. Fill out the **system setting properties**:
        - Set [*Name*] to "`Request number`."
        - Set [*Code*] to "`UsrRequestLastNumber`."

- Select "`Integer`" in the [*Type*] property.
- Set [*Default value*] to "`0`."



Request number

Name* Request number

Type* Integer

Default value 0

Description

Code* UsrRequestLastNumber

Cached ☑

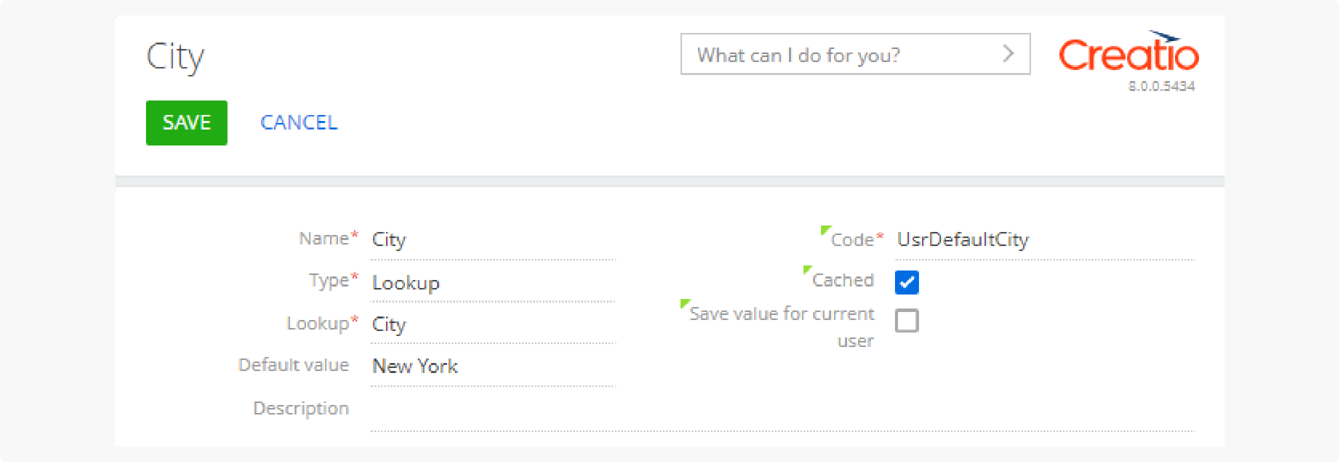
Save value for current user □

3. Add a **system setting that stores the city name**.

   a. Click [*Add setting*] on the section toolbar.

   b. Fill out the **system setting properties**:

   - Set [*Name*] to "`City`."
   - Set [*Code*] to "`UsrDefaultCity`."
   - Select "`Lookup`" in the [*Type*] property.
   - Select "`City`" in the [*Lookup*] property.
   - Select "`New York`" in the [*Default value*] property.



City

Name* City

Type* Lookup

Lookup* City

Default value New York

Description

Code* UsrDefaultCity

Cached ☑

Save value for current user □

4. Go to the `Requests` app page and open the [*Requests form page*] page in the workspace.

   The [*Requests form page*] page includes the [*Name*] field by default.

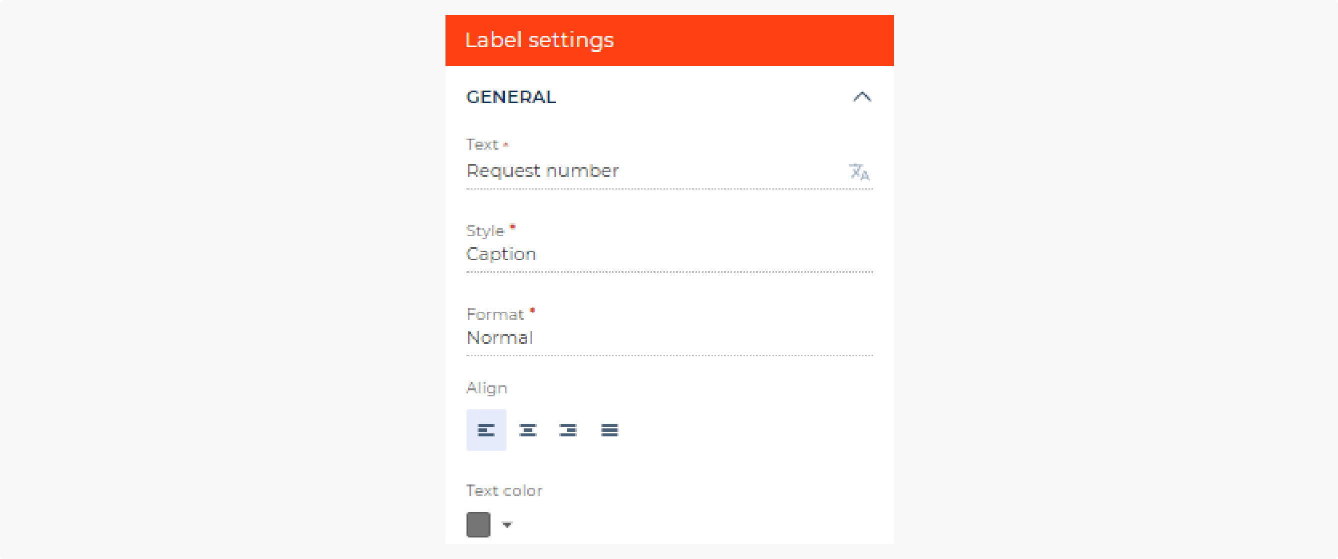
5. Add a **request number label**.

   a. Add a [*Label*] component to the Freedom UI Designer's workspace.

   b. Click the button on the Freedom UI Designer's action panel and fill out the **title's properties** in the setup area:

   - Set [*Text*] to "`Request number`."

- Select "`Caption`" in the [*Style*] property.
- Select the grey color in the [*Text color*] property.



Label settings

GENERAL

Text *
Request number

Style *
Caption

Format *
Normal

Align

Text color

6. Add the following **labels** the same way:

- The value of the system setting that stores the request number.
- Cities.
- The value of the city system setting.

View the label properties to add in the table below.

Label property values

Element	Property	Property value
A label that contains the value of the system setting that stores the request number.	[*Text*]	`"Request number (value)"`
	[*Style*]	Select `"Body text"`
City label	[*Text*]	`"City"`
	[*Style*]	Select `"Caption"`
	[*Text color*]	Select the grey color
A label that contains the value of the city system setting.	[*Text*]	`"City (value)"`
	[*Style*]	Select `"Body text"`

7. Click the </> button on the Freedom UI Designer’s action bar. The source code of the Freedom UI page opens after you save the page settings.

2. Manage system setting values

Configure the business logic in the Client Module Designer. In this example, configure how to manage system settings values.

1. Enable the `sdk.SysSettingsService` **system setting service**. To do this, add the `@creatio-devkit/common` dependency to the AMD module.

AMD module dependencies

```
/* AMD module declaration. */
define("UsrAppRequests_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_DEPS*/
    return {
        ...
    };
});
```

2. Add the following **attributes** to the `viewModelConfig` schema section:

- `UsrDefaultCity`. Stores data about the value of the `UsrDefaultCity` system setting.
- `UsrRequestLastNumber`. Stores data about the value of the `UsrRequestLastNumber` system setting.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that stores the value of the UsrDefaultCity system setting. */
        "UsrDefaultCity": {},
        /* The attribute that stores the value of the UsrRequestLastNumber system setting. */
        "UsrRequestLastNumber": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

3. Change the following **property values** in the `viewConfigDiff` schema section:

- `caption` property for the `RequestNumberValue` element. Bind the property to the value of the `$UsrRequestLastNumber` attribute. The `caption` property is responsible for the text that the element contains.
- `caption` property for the `CityValue` element. Bind the property to the value of the `$UsrDefaultCity` attribute.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "RequestNumberValue",
        "values": {
            ...,
            /* The property responsible for the text that the element contains. Bound to the
            "caption": "$UsrRequestLastNumber",
            ...
        },
        ...
    },
    ...,
    {
        "operation": "insert",
        "name": "CityValue",
        "values": {
            ...,
            /* The property responsible for the text that the element contains. Bound to the
            "caption": "$UsrDefaultCity",
            ...
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

4. Add a custom implementation of the system query handler to the `handlers` schema section. Execute the handler when the `View` model is initialized.

    a. Instantiate the system value service from `@creatio-devkit/common`.

    b. Retrieve the value of the `UsrDefaultCity` system setting and write it to the `UsrDefaultCity` attribute.

    c. Retrieve the page state.

    d. Retrieve the value of the `UsrRequestLastNumber` system setting.

    e. When creating a new record or copying an existing record, send a query to update the value of the `UsrRequestLastNumber` system setting (increase it by 1).

    f. Update the value of the `UsrRequestLastNumber` attribute.

    For Creatio version **8.0.6 and later**

    For Creatio version **8.0-8.0.5**

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

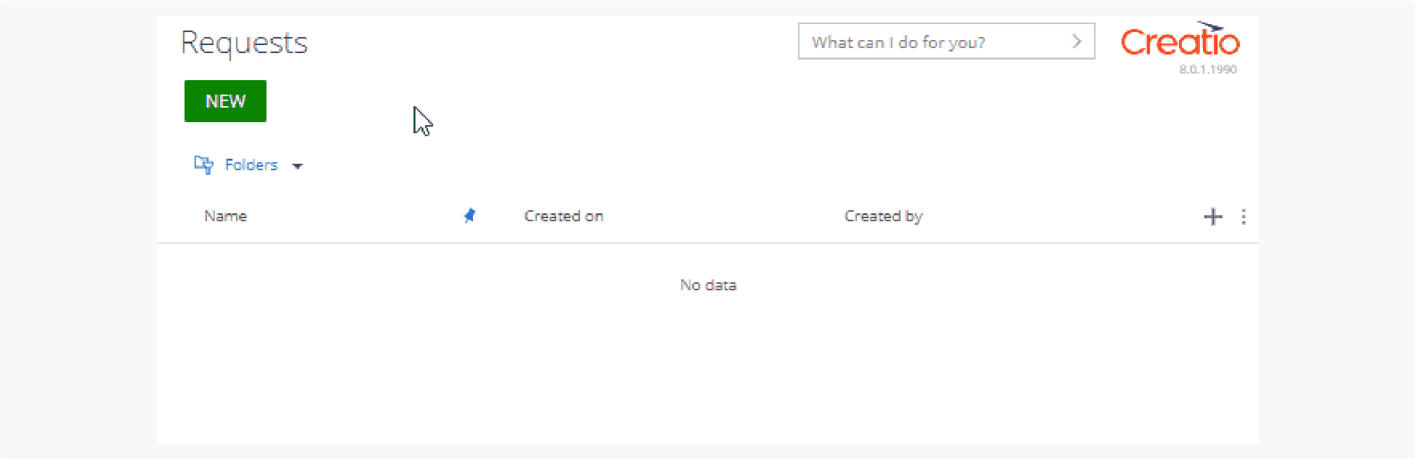
Outcome of the example

To **view the outcome of the example when creating a new request**:

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

As a result, the record page of the [*Requests*] custom section will display:

- User’s city. The default value of the [*City*] (`UsrDefaultCity` code) system setting is used.
- The number of the last created record in the [*Requests*] section, increased by 1. The value of the [*Request number*] (`UsrRequestLastNumber` code) system setting is used. It is increased by 1 when a new request is created.

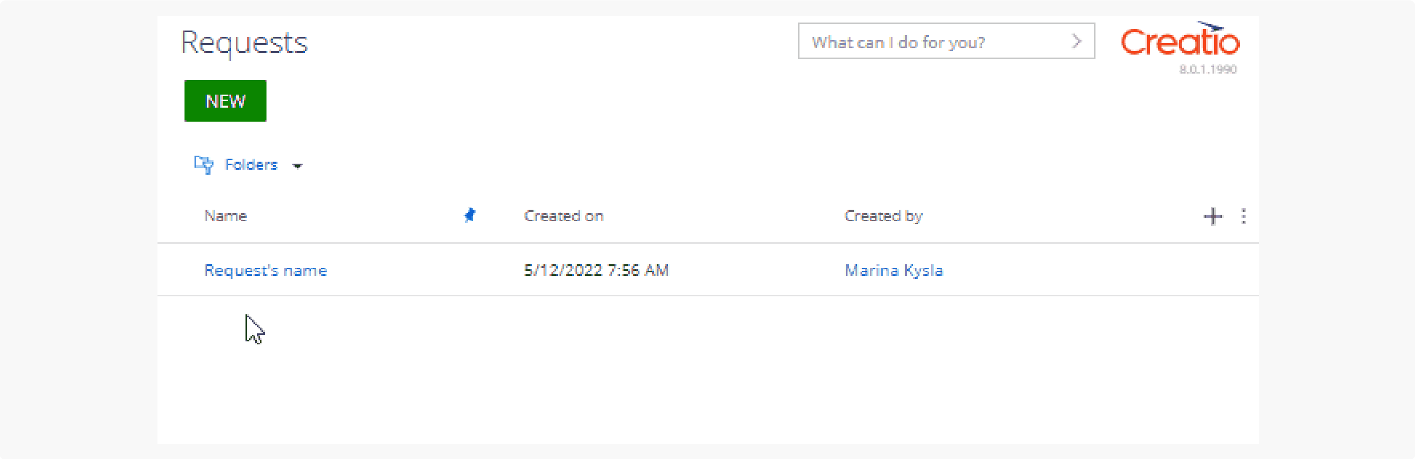


To **view the outcome of the example when copying an existing request**:

1. Add a [*Request's name*] request first.
2. Click ⋮ → [*Copy*] in the [*Request's name*] request's row in the record list.

As a result, the page of the copied record of the [*Requests*] custom section will display:

- User's city. The default value of the [*City*] (`UsrDefaultCity` code) system setting is used.
- The number of the last created record in the [*Requests*] section, increased by 1. The value of the [*Request number*] (`UsrRequestLastNumber` code) system setting is used. It is increased by 1 when an existing request is copied.



Change where the query handler is invoked on the page

Medium

Example. Display the number of the newest section record on the record page of the [*Requests*] custom section. Set the number to the custom system setting. Increment the number by 1 after saving the request page.

1. Set up page UI

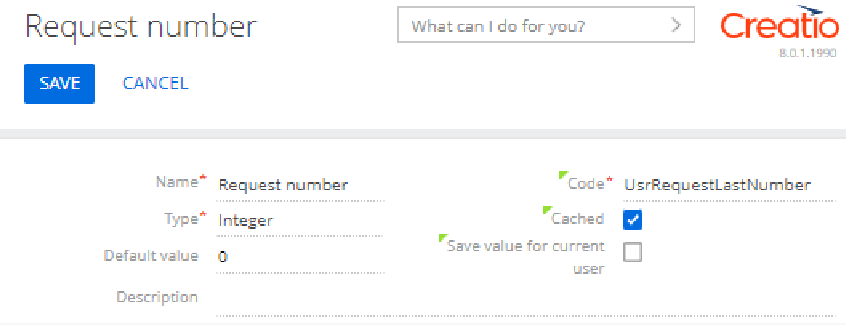
1. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the procedure in the user documentation: [Create a custom app](#).
2. Add **a system setting that stores the request number**.

   a. Click ⚙ to open the System Designer. Click [*System settings*] in the [*System setup*] block.

   b. Click [*Add setting*] on the section toolbar.

   c. Fill out **the system setting properties**:

      - Set [*Name*] to "`Request number`."
      - Set [*Code*] to "`UsrRequestLastNumber`."
      - Select "`Integer`" in the [*Type*] property.
      - Set [*Default value*] to "1."



Name* Request number

Type* Integer

Default value 0

Description

Code* UsrRequestLastNumber

Cached ☑

Save value for current user □

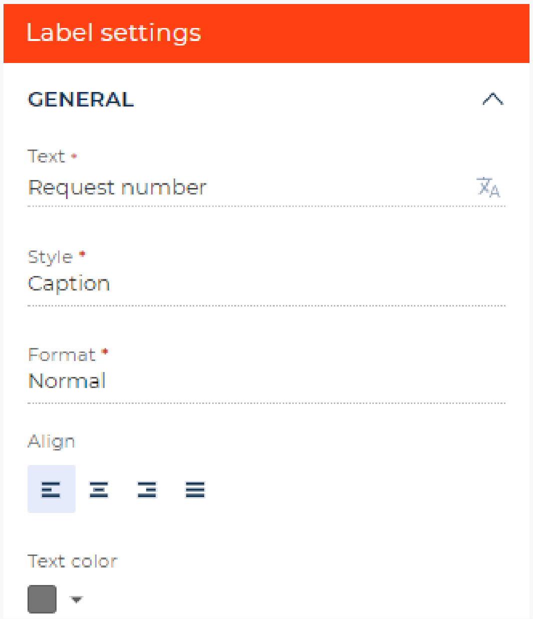
3. Open the [*Requests form page*] page in the workspace of the `Requests` app page.

   The [*Requests form page*] page includes the [*Name*] field by default.
4. Add a [*2 columns*] component to the Freedom UI Designer workspace.
5. Add a **request number label**.

   a. Add a component of the [*Label*] type to the first column of the [*2 columns*] component of the Freedom UI Designer.

   b. Click the ☰ button on the Freedom UI Designer's action panel and fill in the **label's properties** in the setup area:

      - Set [*Title*] to "`Create request`."
      - Select "`Caption`" in the [*Style*] property.
      - Select the gray color in the [*Text color*] property.



Label settings

GENERAL

Text *
Request number

Style *
Caption

Format *
Normal

Align

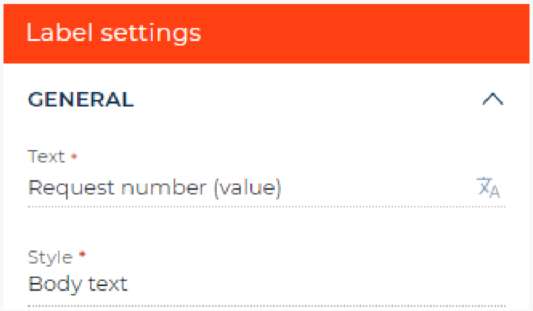
Text color

6. Add a **title that contains the value of the system setting with the request number**.

   a. Add a component of the [*Label*] type to the second column of the [*2 columns*] component of the Freedom UI Designer.

   b. Click the ☰ button on the Freedom UI Designer's action panel and fill in the **title's properties** in the setup area:

   - Set [*Title*] to "`Request number (value)`."
   - Select "`Body text`" in the [*Style*] property.



Label settings

GENERAL

Text *
Request number (value)

Style *
Body text

7. Click the </> button on the Freedom UI Designer's action bar. The source code of the Freedom UI page opens after you save the page settings.

2. Change the origin of the query handler call

Configure the business logic in the Client Module Designer. In this example, change the origin of the query handler call.

1. Enable the `sdk. SysSettingsService` system variable service. To do this, add the `@creatio-devkit/common` dependency to the AMD module.

AMD module dependencies

```
/* AMD module declaration. */
define("UsrAppRequests_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_DEPS*/
    return {
        ...
    };
});
```

2. Add the `UsrRequestLastNumber` attribute to the `viewModelConfig` schema section. The attribute stores the value of the `UsrRequestLastNumber` system setting.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that stores the value of the UsrRequestLastNumber system setting. */
        "UsrRequestLastNumber": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

3. Bind the `caption` property of the `RequestNumberValue` element to the `$UsrRequestLastNumber` model attribute in the `viewConfigDiff` schema section. The `caption` property is responsible for the text that the element contains.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "RequestNumberValue",
        "values": {
            ...,
            /* The property responsible for the text that the element contains. Bound to the
            "caption": "$UsrRequestLastNumber",
            ...
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

4. Add a custom implementation of the `crt.SaveRecordRequest` system query handler to the `handlers` schema section. The handler is called by clicking the [*Save*] button on the request page toolbar.

- Implement waiting for the execution of the next handler. In this example, it is the handler that saves the record.
- Instantiate the system value service from `@creatio-devkit/common` after saving the record.
- Get the value of the `UsrRequestLastNumber` system setting.
- Send a query to update the value of the `UsrRequestLastNumber` system setting (increases it by 1).
- Update the value of the `UsrRequestLastNumber` attribute.
- Return the result of the saving.

`handlers` **schema section**

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "crt.SaveRecordRequest",
        /* Implement the custom query handler. */
        handler: async (request, next) => {
            /* Wait for the next handler (in this case, the handler that saves the record) to
            const saveResult = await next.handle(request);
            /* Instantiate the system value service from @creatio-devkit/common after saving
            const sysSettingsService = new sdk.SysSettingsService();
            /* Retrieve the value of the UsrRequestLastNumber system setting. */
            const requestLastNumber = await sysSettingsService.getByCode('UsrRequestLastNumbe
            /* Send a query to update the value of the UsrRequestLastNumber system setting. *
            await sysSettingsService.update({
                code: 'UsrRequestLastNumber',
                /* The new value is larger than the previous one by 1. */
                value: ++requestLastNumber.value
            });
            /* Update the value of the UsrRequestLastNumber attribute. */
            request.$context.UsrRequestLastNumber = requestLastNumber.value;
            /* Return the result of the saving. */
            return saveResult;
        }
    }
] /**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

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

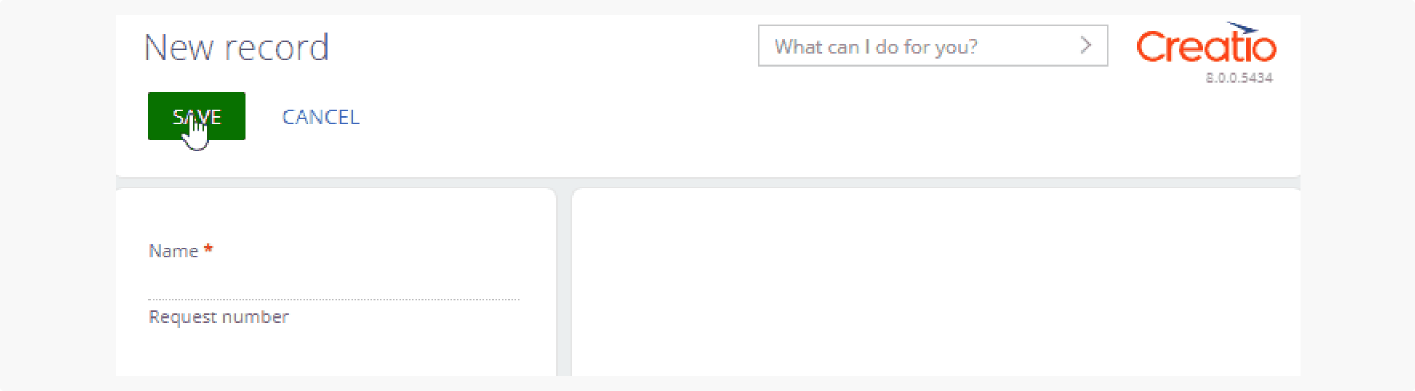
Outcome of the example

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

1. Go to the `Requests` app page and click [*Run app*]

2. Click [*New*] on the `Requests` app toolbar.

As a result, when you click the [*Save*] button on the request page toolbar, Creatio will display the number of the last record created in the [*Requests*] section, increased by 1. The value is taken from the [*Request number*] (`UsrRequestLastNumber` code) system setting.



Customize page fields

Beginner

Creatio 8 Atlas provides the following page customization **actions**:

- Set up a field display condition.
- Set up a condition that locks the field.
- Set up a field population condition.
- Set up a field requirement condition.
- Implement field value validation.
- Implement field value conversion.

Customize the field display condition

1. Add a page field to set the display condition at step 1 of the Freedom UI page customization procedure if needed.
2. Set up the **display condition of the field on the page** at step 2 of the Freedom UI page customization procedure.
   a. Add an attribute that stores data to the `viewModelConfig` schema section.

      Example that adds the `SomeAttributeName` attribute to the client module schema of the Freedom UI page.

      `viewModelConfig` schema section

   b. Bind the `visible` property to the corresponding model attribute in the `viewConfigDiff` schema section. The value of this attribute controls whether the page displays or hides the field. Describe the business logic that changes an attribute value in the `handlers` schema section. The `visible` property is responsible for the visibility of the field.

View an example that binds the `visible` property to the `$SomeAttributeName` attribute below.

`viewConfigDiff` schema section

c. Add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. The handler runs when the value of any attribute changes, including when loading attribute values from a data source. Depending on the attribute value (`true` or `false`), the handler executes different business logic.

   View an example of a `crt.HandleViewModelAttributeChangeRequest` query handler, whose logic depends on the `SomeAttributeName` attribute, below.

   `handlers` schema section

View a detailed example that sets up the field display condition in a separate article: Set up the display condition of a field on a page.

Set up a condition that locks the field

This section covers the procedure for setting up a condition that locks the page field in the app's front-end. To **set up a condition that locks the field in the back-end**, follow the procedure in the user documentation: Access management.

To **set up a condition that locks the field in the front-end**:

1. Add a page field to set up a condition that locks it at step 1 of the Freedom UI page customization procedure if needed.
2. Set up the **condition that locks the field on the page** at step 2 of the Freedom UI page customization procedure.
   a. Add an attribute that stores data to the `viewModelConfig` schema section. Add the attribute the same way as in the setup procedure for the field display condition.
   b. Bind the `readonly` property to the appropriate model attribute in the `viewConfigDiff` schema section. Property binding is similar to that described in the setup procedure for the field display condition. Instead of the `visible` property, use the `readonly` property that locks the field from editing.
   c. Add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. The implementation of the handler is similar to that described in the setup procedure for the field display condition.

View a detailed example that sets up a condition that locks the field in a separate article: Set up the condition that locks the field on a page.

Set up a field population condition

1. Add a page field to configure the population condition at step 1 of the Freedom UI page customization procedure if needed.
2. Set up the **field population condition** at step 2 of the Freedom UI page customization procedure. To do

this, add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. The implementation of the handler is similar to that described in the setup procedure for the field display condition.

View a detailed example that sets up a field population condition in a separate article: Set up the condition that populates a field on a page.

Set up a field requirement condition

1. Add a page field to set the requirement condition at step 1 of the Freedom UI page customization procedure if needed.
2. Set up the **field requirement condition on the page** at step 2 of the Freedom UI page customization procedure.
   a. Bind the `crt.Required` type validator to the model attribute in the `viewModelConfig` schema section. The validator checks that the attribute value is populated.

      View an example that binds a `crt.Required` type validator to a model attribute below.

      `viewModelConfig` schema section

   b. Add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. The handler runs when the value of any attribute changes, including when loading attribute values from a data source. Depending on the value of the attribute (`true` or `false`), the handler executes different business logic.

      View an example of a `crt.HandleViewModelAttributeChangeRequest` query handler, whose logic depends on the `SomeAttributeName` attribute, below.

      `handlers` schema section

View a detailed example that sets up a field requirement condition on a page in a separate article: Set up a field requirement condition on a page.

Implement field value validation

Validators are functions that check whether the value of the `ViewModel` attribute is correct. For example, they can check the value of a record field for compliance with specified conditions. To implement a validator, use the `validators` section of the Freedom UI page schema. Learn more about creating a Freedom UI page in a separate article: Client module.

Creatio applies validators to the `ViewModel` attributes rather than visual elements, but validators can get the validity status data by using `CrtControl`. Validator examples: `MaxLengthValidator`, `MinLengthValidator`, `RequiredValidator`.

To **implement field value validation on the page**:

1. Add a page field whose value to validate at step 1 of the Freedom UI page customization procedure if needed.
2. Implement **field value validation on the page** at step 2 of the Freedom UI page customization procedure.

a. Implement a custom validator in the `validators` schema section.

   The `validators` schema section lets you declare:

   - validator
   - validator function (`function (config)`)
   - validator parameters (`"params"`)
   - whether the validator is asynchronous (`async` flag)

   View an example that declares a custom `usr.SomeValidatorName` validator below.

   `validators` schema section

   `message` is a property that lets you set a custom error message.

f. Bind the validator to an attribute or multiple model attributes by setting different parameters for each of the attributes in the `viewModelConfig` schema section. To do this, specify the `validators` key with the validator's name and its parameters in the corresponding attribute of the `viewModelConfig` schema section.

   View an example that binds the `usr.SomeValidatorName` validator to the `SomeAttributeName1` and `SomeAttributeName2` attributes of the model below.

   `viewModelConfig` schema section

   The priority of the `message` parameter of the attribute configuration object is higher than the priority of the corresponding validator parameter. I. e., for attributes with a `message` parameter set, Creatio generates the error message from the parameter, not from the validator body.

   If an error is caught, the value of the `SomeAttributeName1String` localized string specified in the `SomeAttributeName1` attribute is displayed for the attribute, and the `Some message.` value specified in the validator body is displayed for the `SomeAttributeName2` attribute.

To **disable a validator**, set the `disabled` property of the corresponding validator to `true` (`disabled: true`).

View a detailed example that uses a validator in a separate article: [Implement the field value validation on a page](#).

Implement field value conversion

A **converter** is a function that converts the value of the `ViewModel` attribute bound to the property of the visual component to another value. Converters provided by Creatio 8 Atlas work similarly to Angular filters. Read more in the official [Angular documentation](#). To implement converters, use the `converters` section of the Freedom UI page schema. Learn more about creating a Freedom UI page in a separate article: [Client module](#). Converter examples: `crt.invertBooleanValue`, `crt.toBoolean`.

Converters have the following **special features**:

- applicable only in the `RunTime` mode
- not applicable to constants
- only work in one direction, cannot be used with `CrtControl`

To **implement field value conversion on the page**:

1. Add a page field whose value to convert at step 1 of the Freedom UI page customization procedure if needed.
2. Implement **field value conversion on the page** at step 2 of the Freedom UI page customization procedure.
    a. Implement a custom converter in the `converters` schema section.

       View an example that declares the `usr.SomeConverterName` converter below.

       `converters` schema section

    b. Append the pipe character and the converter type to the name of the attribute to apply the converter in the `viewConfigDiff` schema section.

       View an example that applies the `usr.SomeConverterName` converter to the `$SomeAttributeName` attribute below.

       `viewConfigDiff` schema section

       Besides simple converters, Creatio 8 Atlas provides chains of converters. A **converter chain** comprises multiple converters that are applied to an attribute in a single property.

       View an example that applies a chain of converters (`crt.ToBoolean` and `crt.InvertBooleanValue`) to the `$SomeAttributeName` attribute below.

       `viewConfigDiff` schema section

       Creatio lets you set converter parameters. You can use the same converter several times by setting different parameter values. To **set the converter parameters**, specify the parameter value with the `:` prefix after the converter type. Place the colon character in front of each converter parameter value.

       Available **values of converter parameters**:

       - String. Enclose a string value in single quotes.
       - Number.
       - `true` or `false`.
       - A binding to another attribute.

       View an example that applies the `exmpl.Concat` converter with a `SomeParameter` string parameter to the `SomeAttributeName` attribute below. Note that `exmpl.Concat` is an example converter and is not available for solving actual business problems.

       `viewConfigDiff` schema section

Converters are not available for the following **binding types**:

- binding to resource attribute
- function binding
- binding an event to a model method

View a detailed example that uses a converter in a separate article: Implement field value conversion on a page.

Set up the display condition of a field on a page

Medium

Example. Set up the display condition for the [*Sick leave, days left*] field on the record page of the custom [*Requests*] section. Display the field for requests that originate from employees, i. e., requests whose [*Originator type*] field is set to "`Employee`."

1. Set up the page UI

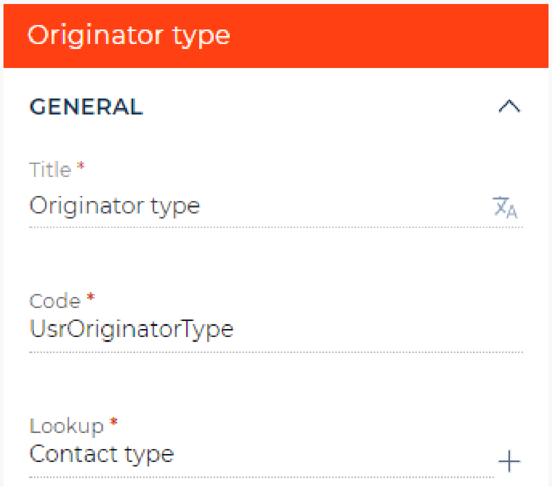
1. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
2. Open the [*Requests form page*] page in the working area of the `Requests` app page.

   The [*Requests form page*] page includes the [*Name*] field by default.
3. Add a **field that contains the request originator type**.

   a. Add a [*Dropdown*] type field to the working area of the Freedom UI Designer.

   b. Click ☰ in the action panel of the Freedom UI Designer and fill out the **field properties** in the setup area.

   - Set [*Title*] to "`Originator type`."
   - Set [*Code*] to "`UsrOriginatorType`."
   - Select "`Contact type`" in the [*Lookup*] property.



Originator type

GENERAL

Title *
Originator type

Code *
UsrOriginatorType

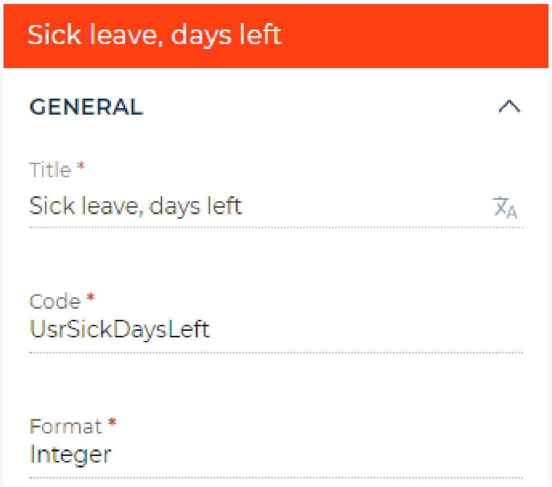
Lookup *
Contact type

4. Add a **field that contains the number of sick days left**.

   a. Add a [*Number*] type field to the working area of the Freedom UI Designer.

b. Click ☰ in the action panel of the Freedom UI Designer and fill out the **field properties** in the setup area.

- Set [*Title*] to "`Sick leave, days left`."
- Set [*Code*] to "`UsrSickDaysLeft`."



Sick leave, days left

GENERAL

Title *
Sick leave, days left

Code *
UsrSickDaysLeft

Format *
Integer

5. Click </> 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.

2. Set up the field display condition

Configure the business logic in the Client Module Designer. For this example, set up the field display condition.

1. Add an `IsRequestFromEmployee` attribute that stores data about the contact type from which the request originates to the `viewModelConfig` schema section.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that stores the request originator type. */
        "IsRequestFromEmployee": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

2. Bind the `visible` property of the `UsrSickDaysLeft` element to the `IsRequestFromEmployee` model attribute in the `viewConfigDiff` schema section. If the request originates from an [*Employee*] type contact, display the [*Sick leave, days left*] field. Hide the field for other contact types.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "UsrSickDaysLeft",
        "values": {
            ...,
            /* The property that flags the field as visible. Bound to the IsRequestFromEmploy
            "visible": "$IsRequestFromEmployee"
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

3. Add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. Run the handler when the value of any attribute changes, including changes made after loading the attribute values from the data source. The handler checks the `UsrOriginatorType` attribute value. If the new attribute value refers to the "`Employee`" value of the [*Contact type*] lookup, set the `IsRequestFromEmployee` attribute value to `true`, otherwise set it to `false`. The unique ID of the [*Employee*] type contact set as the `employeeOriginatorTypeId` constant is stored in the corresponding string of the [*Contact type*] lookup record. In this example, the ID of the [*Employee*] type contact is "`60733efc-f36b-1410-a883-16d83cab0980`."

`handlers` schema section

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "crt.HandleViewModelAttributeChangeRequest",
        /* The custom implementation of the system query handler. */
        handler: async (request, next) => {
            /* Check the request originator type. */
            if (request.attributeName === 'UsrOriginatorType') {
                const employeeOriginatorTypeId = '60733efc-f36b-1410-a883-16d83cab0980';
                const selectedOriginatorType = await request.$context.UsrOriginatorType;
                const selectedOriginatorTypeId = selectedOriginatorType?.value;
                /* If the request originates from an employee, set the IsRequestFromEmployee
                request.$context.IsRequestFromEmployee = selectedOriginatorTypeId === employe
            }
            /* Call the next handler (if it exists) and return its result. */
            return next?.handle(request);
        }
    }
]/**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

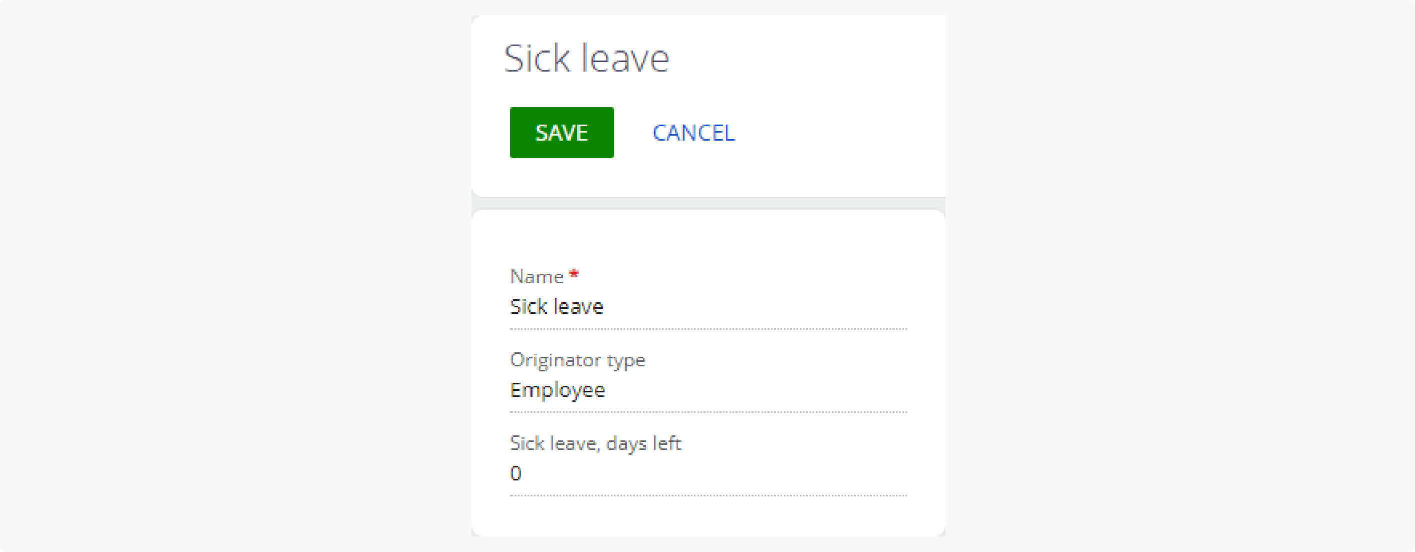
4. Click [*Save*] on the Client Module Designer's toolbar.

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 "`Sick leave`" in the [*Name*] field.
4. Select "`Employee`" in the [*Originator type*] field.

As a result, Creatio will display the [*Sick leave, days left*] field on the page of the request that originates from an [*Employee*] type contact.



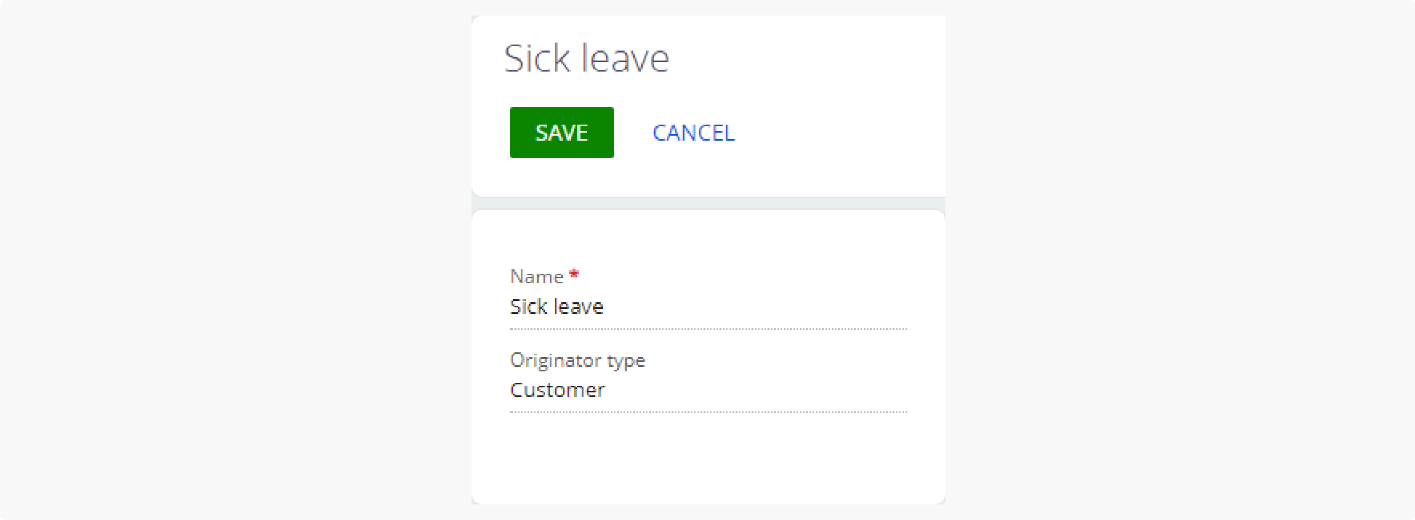
Sick leave

Name *
Sick leave

Originator type
Employee

Sick leave, days left
0

Creatio will not display the [*Sick leave, days left*] field for requests that originate from other contact types, e. g., [*Customer*].



Sick leave

Name *
Sick leave

Originator type
Customer

Set up the condition that locks the field on a page

Medium

This example is implemented in the front-end. Learn more about the back-end implementation in the user documentation guide: Access management.

Example. Set up the condition that locks the [*Applicant*] field on the record page of the custom [*Requests*] section. Lock the field if the request is completed, i. e., the [*Status*] field is set to "`Completed`."

1. Set up the page UI

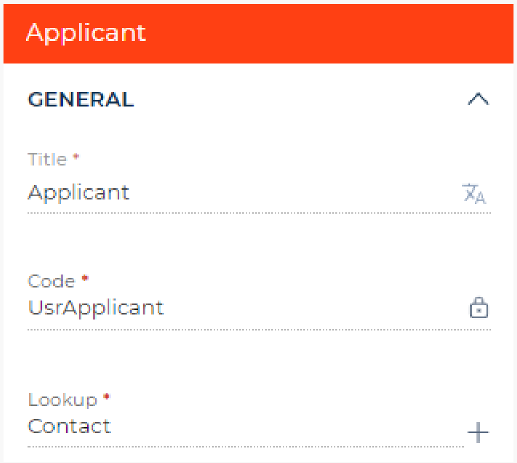
1. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
2. Open the [*Requests form page*] page in the working area of the `Requests` app page.

   The [*Requests form page*] page includes the [*Name*] field by default.
3. Add an **applicant field**.

   a. Add a [*Dropdown*] type field to the working area of the Freedom UI Designer.

   b. Click ⚙ in the action panel of the Freedom UI Designer and fill out the **field properties** in the setup area.

   - Set [*Title*] to "`Applicant`."
   - Set [*Code*] to "`UsrApplicant`."
   - Select "`Contact` " in the [*Lookup*] property.



Applicant

GENERAL

Title *
Applicant

Code *
UsrApplicant

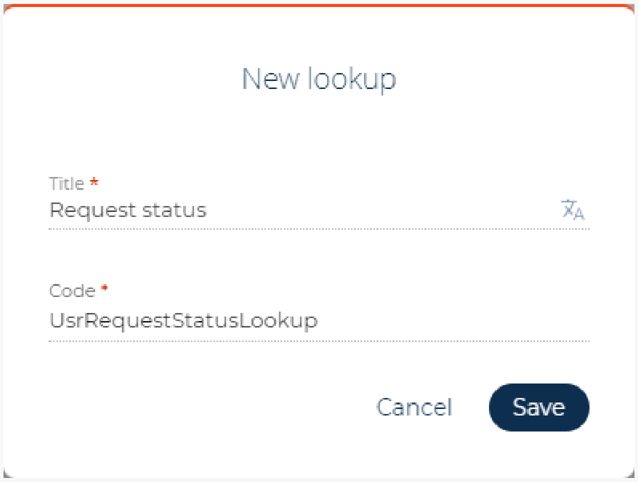
Lookup *
Contact

4. Add a **request status field**.

a. Add a [*Dropdown*] type field to the working area of the Freedom UI Designer.

b. Click ☰ in the action panel of the Freedom UI Designer and fill out the **field properties** in the setup area.

- Set [*Title*] "`Status`."
- Set [*Code*] to "`UsrStatus`."
- Click the + button next to the [*Lookup*] property and fill out the **lookup properties**:
  - Set [*Title*] to "`Request status`."
  - Set [*Code*] to "`UsrRequestStatusLookup`."

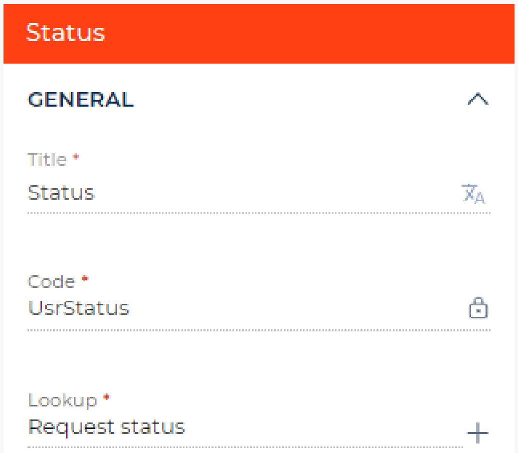


New lookup

Title *
Request status

Code *
UsrRequestStatusLookup

Click [*Save*] to add the lookup.



Status

GENERAL

Title *
Status

Code *
UsrStatus

Lookup *
Request status

h. Click [*Save*] on the Freedom UI Designer's toolbar.

5. Fill out the [*Request status*] **lookup**.

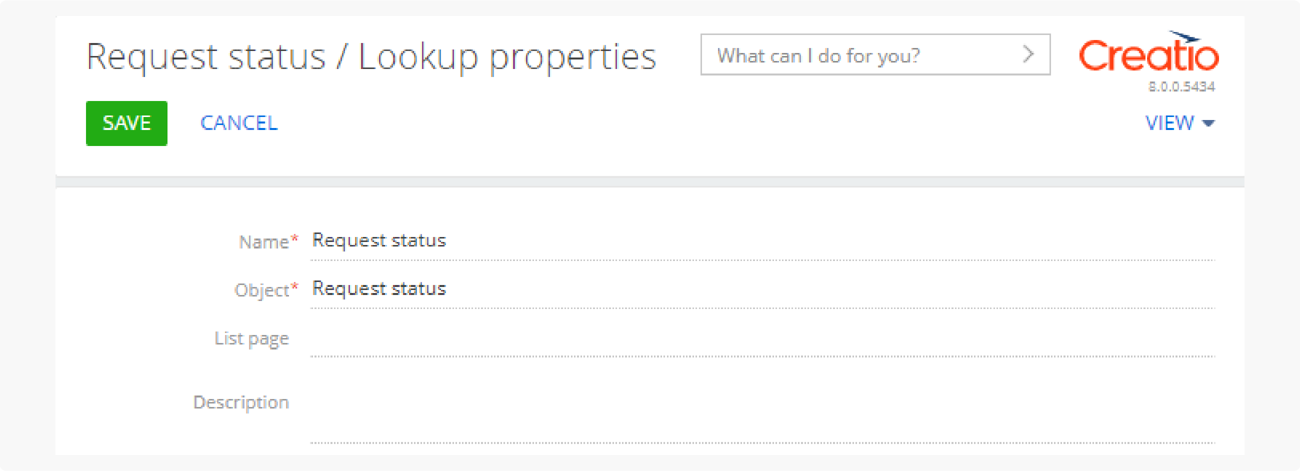
a. Open the `Requests` app page and click [*Run app*].

b. Click ⚙ to open the System Designer. Go to the [*System setup*] block → [*Lookups*].

c. If you use Creatio 8.0.0, register the **lookup**. The lookup is registered automatically in Creatio 8.0.1 and later.

   a. Click [*New lookup*] on the [*Lookups*] section toolbar and fill out the **lookup properties**:

      - Set [*Name*] to "`Request status`."
      - Select "`Request status` " in the [*Object*] property.



Name* Request status

Object* Request status

List page

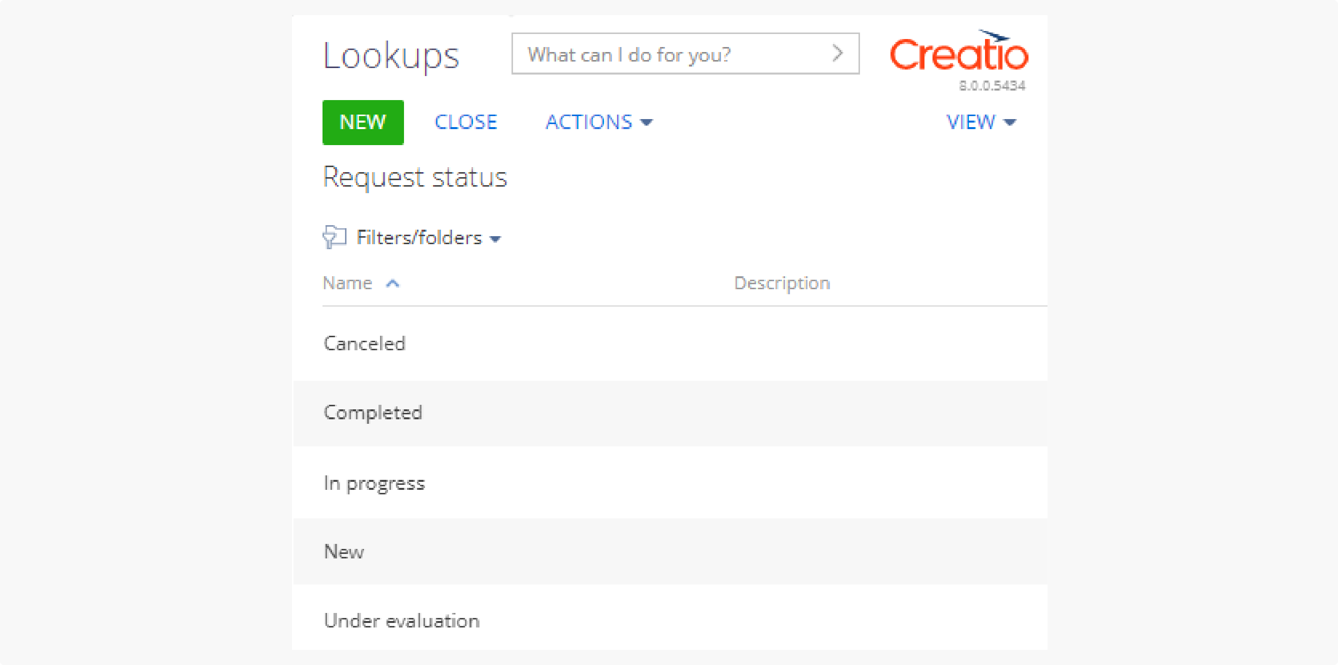
Description

   d. Click [*Save*] on the lookup setup page's toolbar.

d. Open the [*Request status*] toolbar.

e. Click [*New*] on the lookup setup page's toolbar and add the following **lookup values**:

- "`New`"
- "`Under evaluation`"
- "`In progress`"
- "`Canceled`"
- "`Completed`"



Canceled

Completed

In progress

New

Under evaluation

6. Open the [*Requests form page*] page and click the </> button on the Freedom UI Designer's toolbar.

   After you save the page settings, Creatio opens the source code of the Freedom UI page.

2. Set up the condition that locks the field

Configure the business logic in the Client Module Designer. For this example, set up the condition that locks the field.

1. Add an `IsApplicantReadonly` attribute that stores data about the contact's permission to edit the [*Applicant*] field to the `viewModelConfig` schema section.

   `viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that locks the [Applicant] field. */
        "IsApplicantReadonly": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

2. Bind the `readonly` property of the `UsrApplicant` element to the `IsApplicantReadonly` model attribute in the `viewConfigDiff` schema section. Lock the [*Applicant*] field if the request is completed. Keep the field editable for other request statuses.

   `viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "UsrApplicant",
        "values": {
            ...,
            /* The property that locks the field from editing. Bound to the IsApplicantReadon
            "readonly": "$IsApplicantReadonly"
        },
        ...
    },
    ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

3. Add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. Run the handler when the value of any attribute changes, including changes made after loading the attribute values from the data source. The handler checks the `UsrStatus` attribute value. If the new attribute value refers to the "`Completed`" value of the [*Request status*] lookup, set the `IsApplicantReadonly` attribute value to `true`, otherwise set it to `false`. The unique status ID of the completed request set as the `completedStatusId` constant is stored in the corresponding column of the [*Request status*] lookup's record string. To display the [*Id*] column in the [*Request status*] lookup list, follow the guide in the user documentation: [Work with record lists](). In this example, the status ID of the completed request is "`6d76b4e0-6507-4c34-902b-90e18df84153`."

`handlers` **schema section**

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "crt.HandleViewModelAttributeChangeRequest",
        /* The custom implementation of the system query handler. */
        handler: async (request, next) => {
            /* Check the request status. */
            if (request.attributeName === 'UsrStatus') {
                const completedStatusId = '6d76b4e0-6507-4c34-902b-90e18df84153';
                const selectedStatus = await request.$context.UsrStatus;
                const selectedStatusId = selectedStatus?.value;
                const isRequestCompleted = selectedStatusId === completedStatusId;
                /* If the request status is [Completed], set the IsApplicantReadonly attribut
                request.$context.IsApplicantReadonly = isRequestCompleted;
            }
            /* Call the next handler (if it exists) and return its results. */
            return next?.handle(request);
        }
    }
]/**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

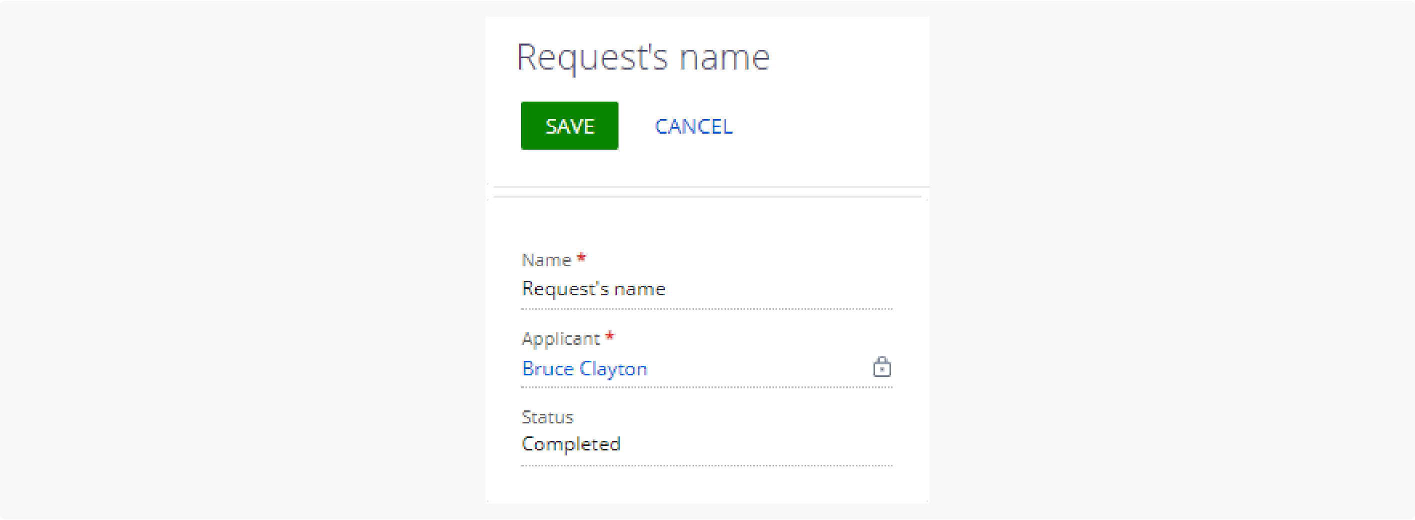
4. Click [*Save*] on the Client Module Designer's toolbar.

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 "`Request's name`" in the [*Name*] field.
4. Select "`Bruce Clayton`" in the [*Applicant*] field.
5. Select "`Completed`" in the [*Status*] field.

As a result, Creatio will lock the [*Applicant*] field for completed requests.



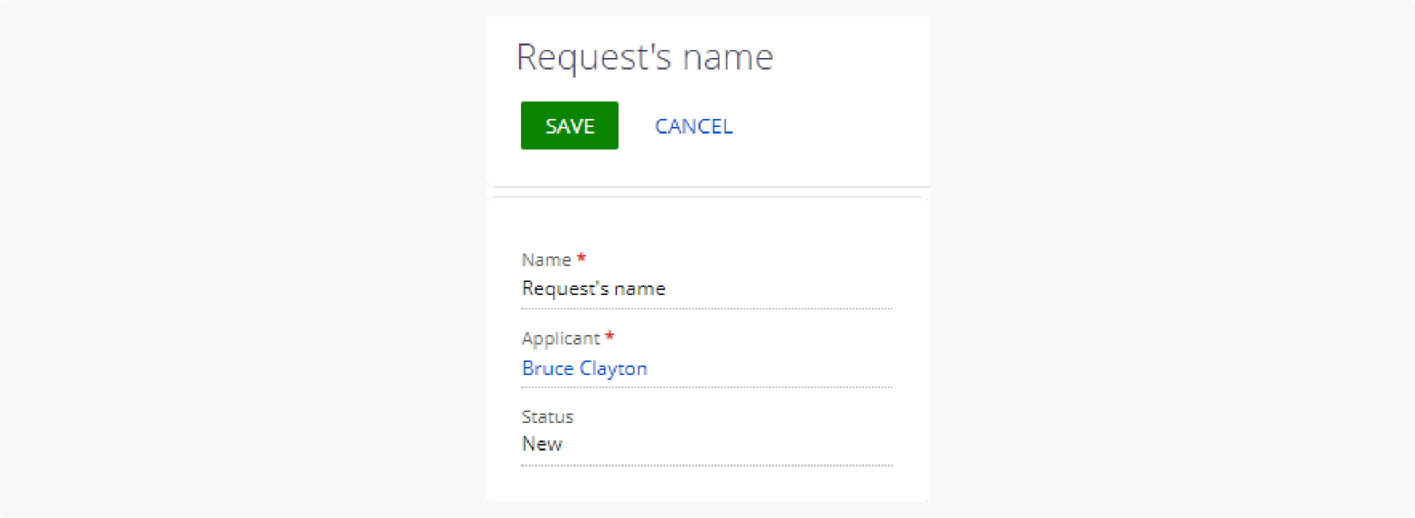
Request's name

Name *
Request's name

Applicant *
Bruce Clayton 🔒

Status
Completed

The [*Applicant*] field will be editable for requests that have a different status. For example, "`New`."



Request's name

Name *
Request's name

Applicant *
Bruce Clayton

Status
New

Set up the condition that populates a field on a page

Medium

Example. Set up the condition that populates the [*Description*] field on the record page of the custom [*Requests*] section. If the [*Name*] and [*Description*] field values match, populate the [*Description*] field with the new [*Name*] field value. Otherwise, leave the [*Description*] field value as is.

1. Set up the page UI

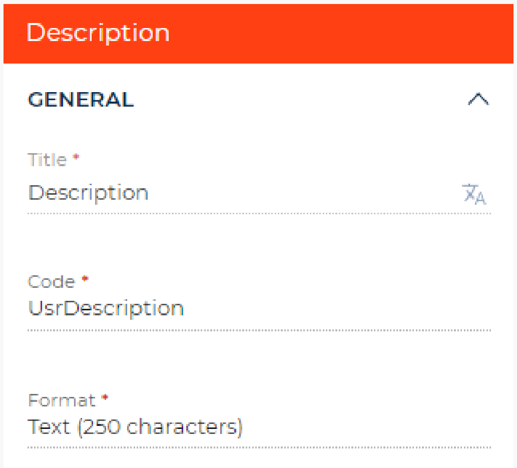
1. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
2. Open the [*Requests form page*] page in the working area of the `Requests` app page.

   The [*Requests form page*] page includes the [*Name*] field by default.
3. Add a **request description field**.

   a. Add a [*Text*] type field to the working area of the Freedom UI Designer.

   b. Click ☰ in the action panel of the Freedom UI Designer and fill out the **field properties** in the setup area.

   - Set [*Title*] to "`Description`."
   - Set [*Code*] to "`UsrDescription`."



Description

GENERAL

Title *
Description

Code *
UsrDescription

Format *
Text (250 characters)

4. Click </> 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.

2. Set up the condition that populates the field

Configure the business logic in the Client Module Designer. For this example, set up the condition that populates the field.

1. Add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. Run the handler when the value of any attribute changes, including changes made after loading the attribute values from the data source. The handler checks the `UsrName` attribute value. If the old attribute value matches the `UsrDescription` attribute value, set the `UsrDescription` attribute to the same value as the new `UsrName` attribute value.

   `handlers` schema section

   ```
   handlers: /**SCHEMA_HANDLERS*/[
       {
           request: "crt.HandleViewModelAttributeChangeRequest",
           /* The custom implementation of the system query handler. */
           handler: async (request, next) => {
               /* If the UsrName field changes, take the following steps. */
               if (request.attributeName === 'UsrName') {
                   /* Check whether the old UsrName field value matches the UsrDescription field
                   const isFieldsShouldBeSynchronized = request.oldValue ===   await request.$co
                   if (isFieldsShouldBeSynchronized) {
                       /* Assign the new UsrName field value to the UsrDescription field. */
                       request.$context.UsrDescription = await request.$context.UsrName;
                   }
               }
               /* Call the next handler if it exists and return its result. */
               return next?.handle(request);
           }
       }
   ]/**SCHEMA_HANDLERS*/,
   ```

   Complete source code of the page schema

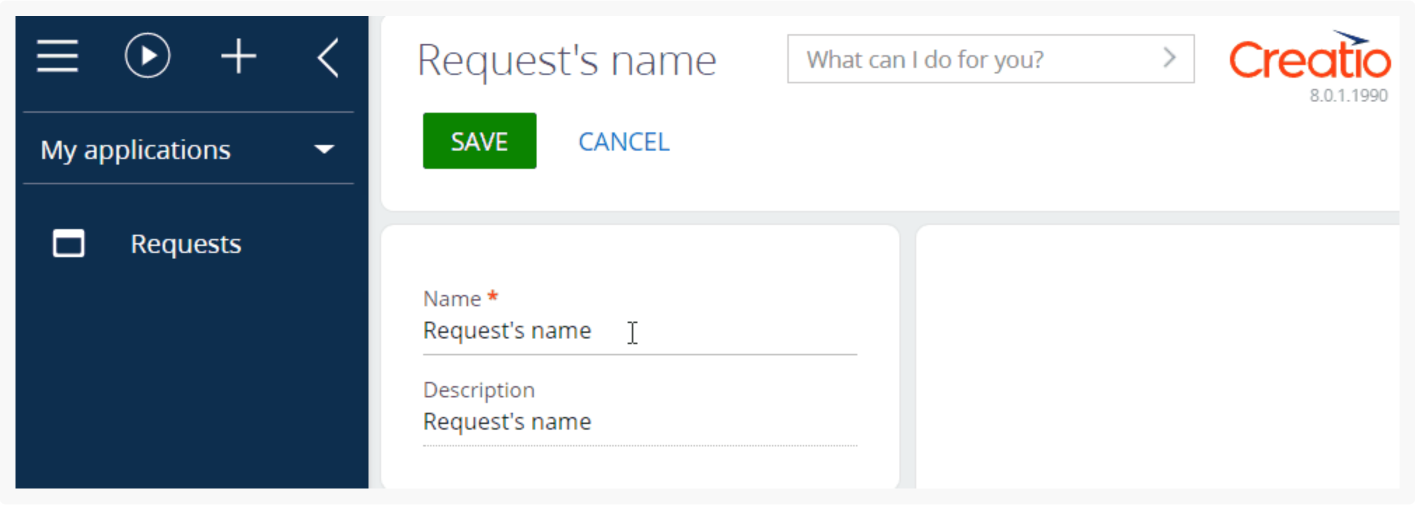
2. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

To **view the outcome of the example for the same** [*Name*] and [*Description*] field values:

1. Open the `Requests` app page and click [*Run app*].
2. Click [*New*] on the `Requests` app toolbar.
3. Enter "`Request's name`" in the [*Name*] field.
4. Enter "`Request's name`" in the [*Description*] field.
5. Change the [*Name*] field value to "`Test`."

As a result, Creatio will set the [*Description*] field to the "`Test`," same as the [*Name*] field.



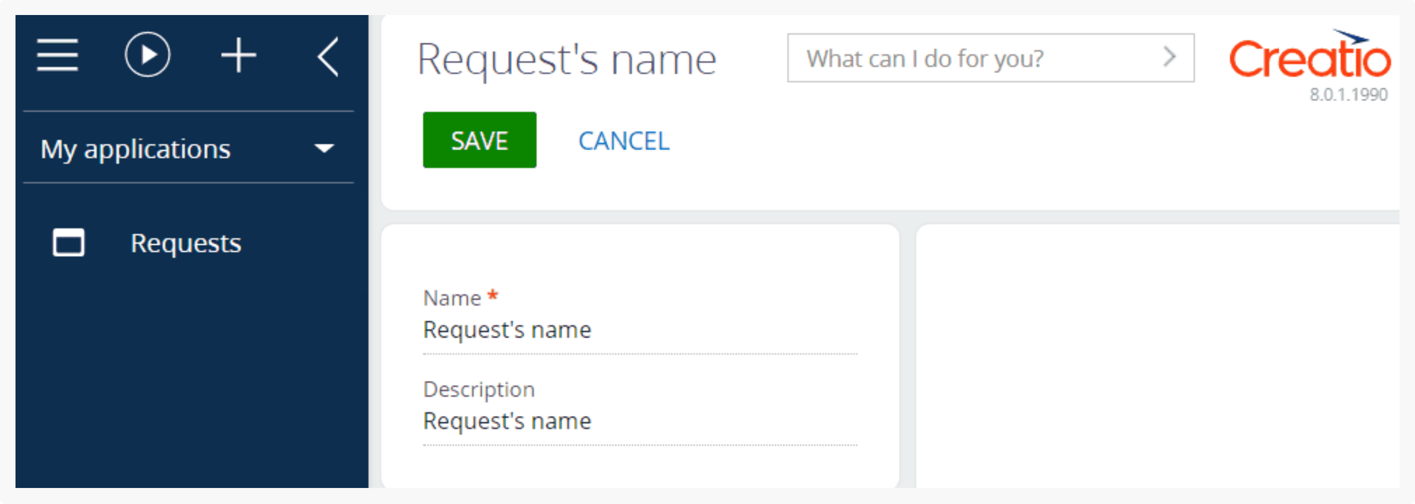
Name *
Request's name

Description
Request's name

To **view the outcome of the example for different** [*Name*] and [*Description*] field values:

1. Change the [*Description*] field value to "`Request's description`."
2. Enter "`Test`" in the [*Name*] field.

As a result, Creatio will leave the [*Description*] field value as is.



Name *
Request's name

Description
Request's name

Set up the requirement condition of a field on a page

Medium

Example. Make the [*Description*] field on the record page of the custom [*Requests*] section required. The field must be required if the request is new, i. e., the [*Status*] field is set to "`New`."

1. Set up the page UI

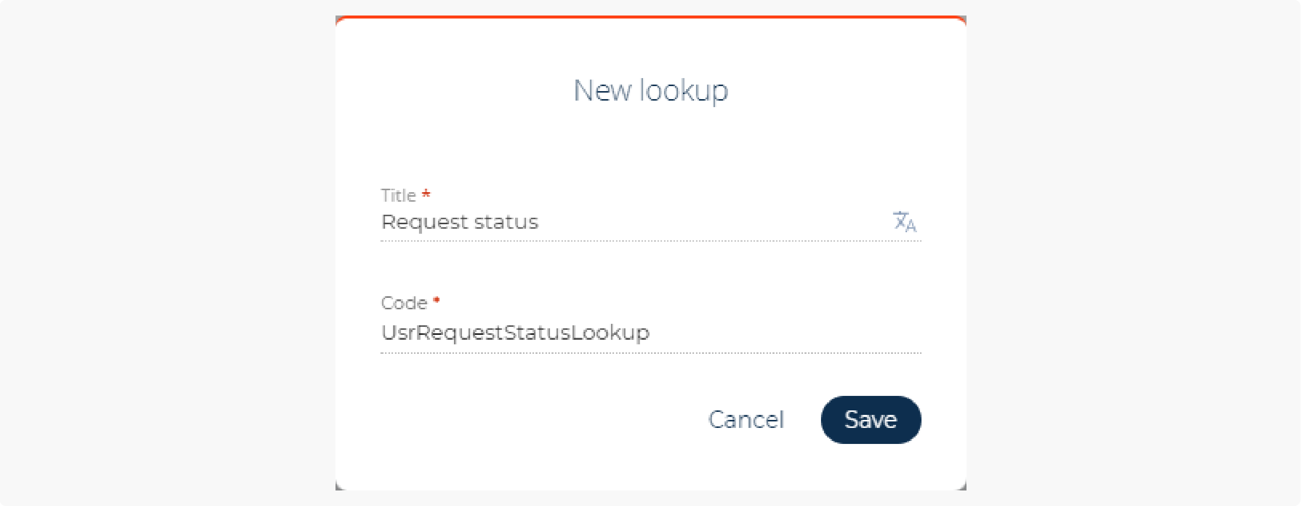
1. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: [Create a custom app](#).
2. Open the [*Requests form page*] page in the working area of the `Requests` app page.

   The [*Requests form page*] page includes the [*Name*] field by default.
3. Add a **request status field**.

   a. Add a [*Dropdown*] type field to the working area of the Freedom UI Designer.

   b. Click in the action panel of the Freedom UI Designer and fill out the **field properties** in the setup area.

   - Set [*Title*] "`Status`."
   - Set [*Code*] to "`UsrStatus`."
   - Click the + button next to the [*Lookup*] property and fill out the **lookup properties**:
     - Set [*Title*] to "`Request status`."
     - Set [*Code*] to "`UsrRequestStatusLookup`."

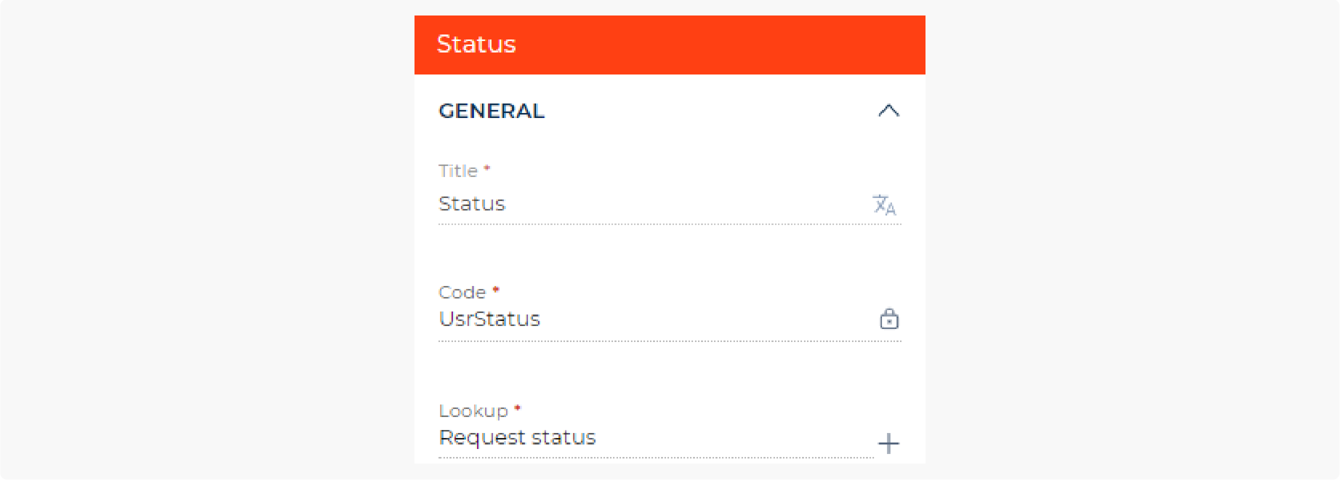


New lookup

Title * Request status

Code * UsrRequestStatusLookup

Click [*Save*] to add the lookup.



Status

GENERAL

Title * Status

Code * UsrStatus

Lookup * Request status

h. Click [*Save*] on the Freedom UI Designer's toolbar.

4. Fill out the [*Request status*] **lookup**.

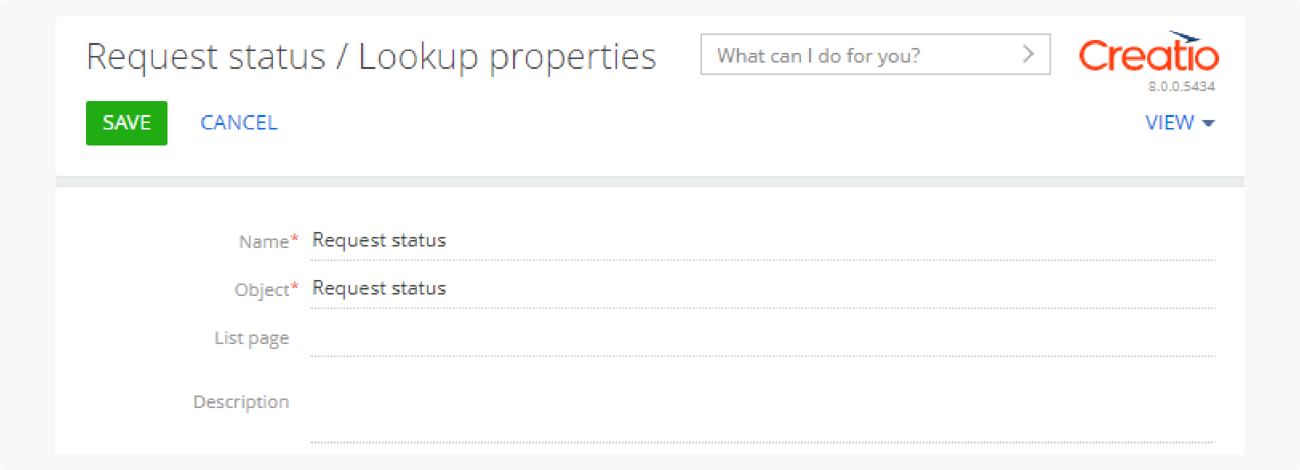
   a. Open the `Requests` app page and click [*Run app*].

   b. Click ⚙ to open the System Designer. Go to the [*System setup*] block → [*Lookups*].

   c. If you use Creatio 8.0.0, register the **lookup**. The lookup is registered automatically in Creatio 8.0.1 and later.

      a. Click [*New lookup*] on the [*Lookups*] section toolbar and fill out the **lookup properties**:

      - Set [*Name*] to "`Request status`."
      - Set [*Object*] to "`Request status`."



Name* Request status

Object* Request status

List page

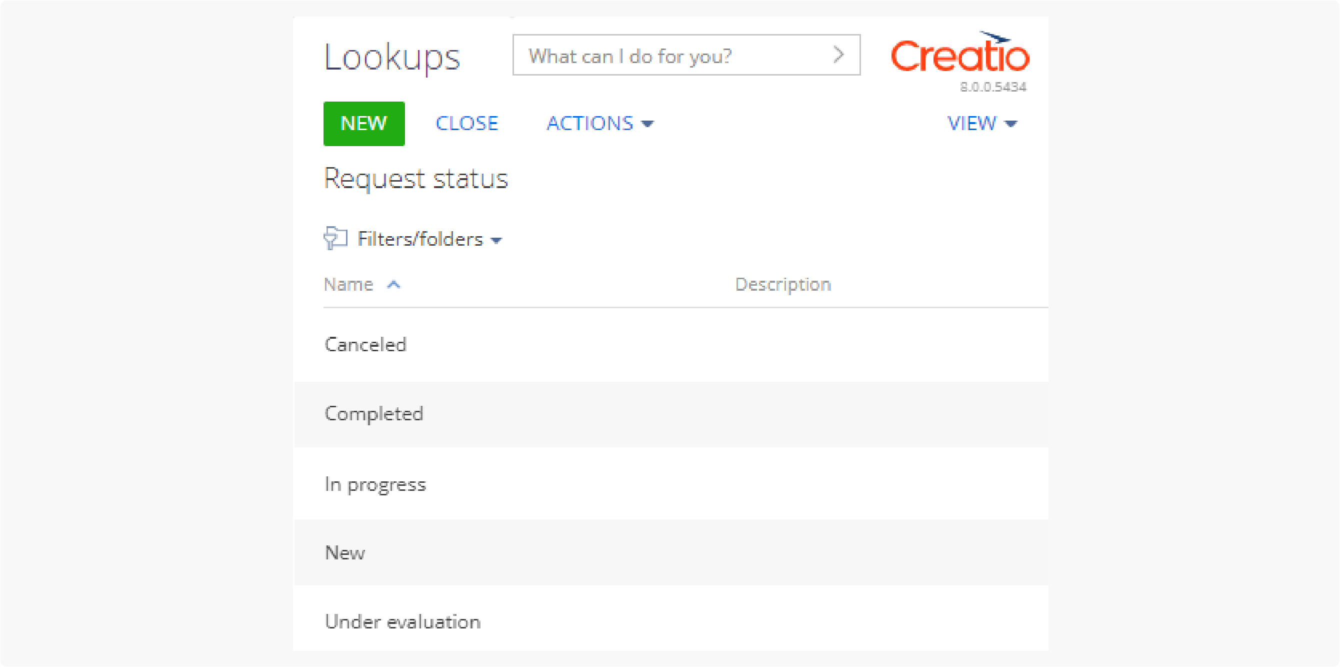
Description

      d. Click [*Save*] on the lookup setup page's toolbar.

   d. Open the [*Request status*] toolbar.

   e. Click [*New*] on the lookup setup page's toolbar and add the following **lookup values**:

   - "`New`"
   - "`Under evaluation`"
   - "`In progress`"
   - "`Canceled`"
   - "`Completed`"



Request status

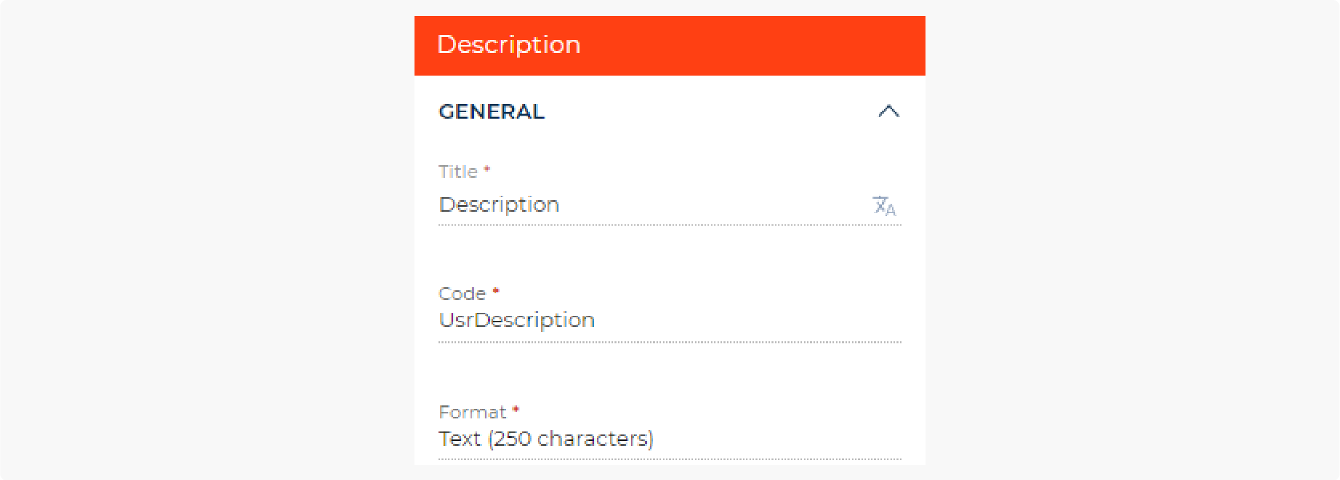
Name	Description
Canceled	
Completed	
In progress	
New	
Under evaluation	

5. Add a **request description field**.

   a. Open the [*Requests form page*] page and add a [*Text*] type field to the working area of the Freedom UI Designer.

   b. Click in the action panel of the Freedom UI Designer and fill out the **field properties** in the setup area.

   - Set [*Title*] to "`Description`."
   - Set [*Code*] to "`UsrDescription`."



Title *
Description

Code *
UsrDescription

Format *
Text (250 characters)

6. Click 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.

2. Set up the condition that makes the field required

Configure the business logic in the Client Module Designer. For this example, set up the condition that makes the field required.

1. Bind the `crt.Required` type validator to the `UsrDescription` model attribute in the `viewModelConfig` schema section. The validator checks the attribute value.

   `viewModelConfig` schema section

   ```
   viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
       "attributes": {
           ...
           "UsrDescription": {
               ...,
               /* The property that enables validators in the attribute. */
               "validators": {
                   /* Flag the field as required. */
                   "required": {
                       "type": "crt.Required"
                   }
               }
           },
           ...
       }
   }/**SCHEMA_VIEW_MODEL_CONFIG*/,
   ```

2. Add a custom implementation of the `crt.HandleViewModelAttributeChangeRequest` system query handler to the `handlers` schema section. Run the handler when the value of any attribute changes, including changes made after loading the attribute values from the data source. The handler checks the `UsrStatus` attribute value. If the new attribute value refers to the "`New`" value of the [*Request status*] lookup, apply the validator, otherwise do not apply it. The unique status ID of the new request set as the `newStatusId` constant is stored in the corresponding column of the [*Request status*] lookup's record string. To display the [*Id*] column in the [*Request status*] lookup list, follow the guide in the user documentation: Work with record lists. In this example, the status ID of the new request is "`3be636fa-12b4-40eb-a050-91b1d774a75f`."

   `handlers` schema section

   ```
   handlers: /**SCHEMA_HANDLERS*/[
       {
           request: "crt.HandleViewModelAttributeChangeRequest",
           /* The custom implementation of the system query handler. */
           handler: async (request, next) => {
               if (request.attributeName === 'UsrStatus') {
                   const newStatusId = '3be636fa-12b4-40eb-a050-91b1d774a75f';
                   const selectedStatus = await request.$context.UsrStatus;
                   const selectedStatusId = selectedStatus?.value;
                   const isNewRequest = selectedStatusId === newStatusId;
                   /* Check the request status. */
                   if (isNewRequest) {
   ```

```
                    /* If the request is new, apply the required validator to the UsrDescript
                    request.$context.enableAttributeValidator('UsrDescription', 'required');
                } else {
                    /* Do not apply the required validator to the UsrDescription attribute fo
                    request.$context.disableAttributeValidator('UsrDescription', 'required');
                }
            }
            /* Call the next handler if it exists and return its result. */
            return next?.handle(request);
        }
    }
]/**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

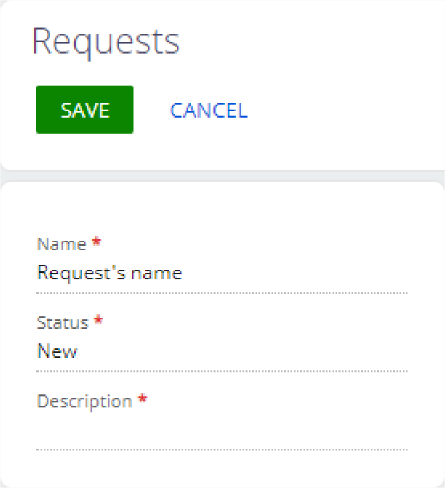
3. Click [*Save*] on the Client Module Designer's toolbar.

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 "`Request's name`" in the [*Name*] field.
4. Select "`New`" in the [*Status*] field.

As a result, Creatio will make the [*Description*] field required for new requests.

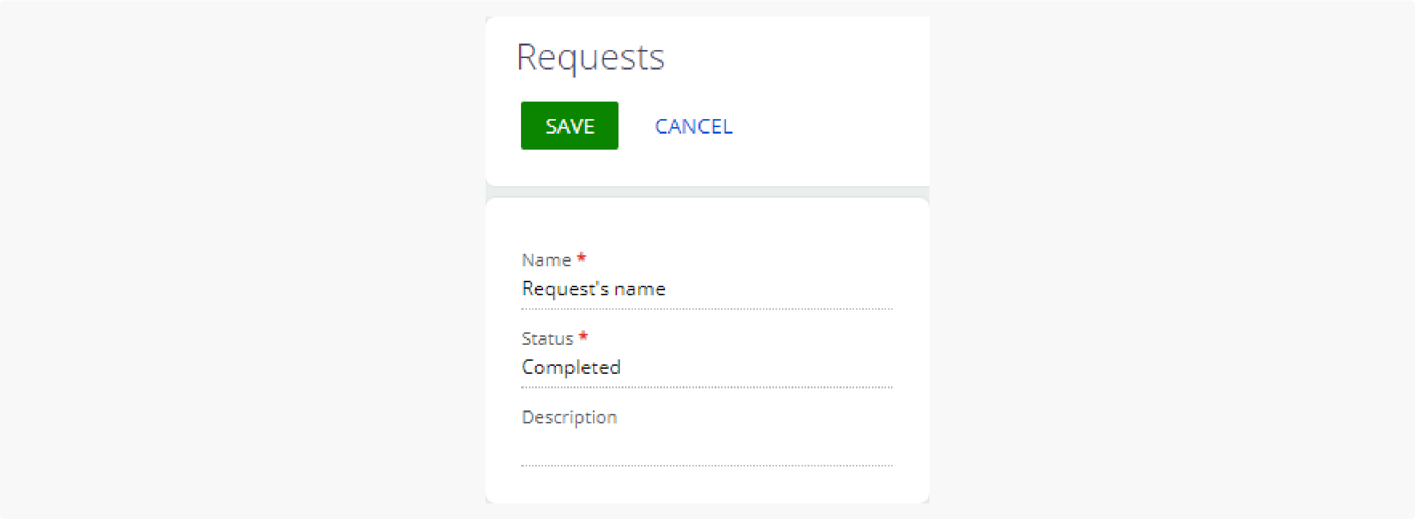


Name *
Request's name

Status *
New

Description *

The [*Description*] field will not be required for requests that have other statuses. For example, "`Completed`."


Requests

Name *
Request's name

Status *
Completed

Description

Implement the field value validation on a page

Medium

Example. Add a validator that ensures the [*Name*] field does not contain the `test` value to the record page of the custom [*Validators*] section.

1. Set up the page UI

1. Create a custom `Validators` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
2. Open the [*Validators form page*] page in the working area of the `Validators` app page.

   The [*Validators form page*] page includes the [*Name*] field by default.
3. Click </> 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.

2. Set up the field validation

Configure the business logic in the Client Module Designer. For this example, set up the field validation. Add a validator to the [*Name*] field of the [*Validators form page*] page.

1. Implement a custom `usr.MyValidator` validator in the `validators` schema section.

`validators` schema section

```
validators: /**SCHEMA_VALIDATORS*/{
    /* The validator type must contain a vendor prefix.
    Format the validator type in PascalCase. */
    "usr.MyValidator": {
        "validator": function (config) {
            return function (control) {
                return control.value !== config.invalidName ? null: {
                    "usr.MyValidator": { message: config.message }
                };
            };
        },
        "params": [
            {
                "name": "invalidName"
            },
            {
                "name": "message"
            }
        ],
        "async": false
    }
}/**SCHEMA_VALIDATORS*/
```

2. Bind the `MyValidator` validator to the `UsrName` model attribute in the `viewModelConfig` schema section. Specify the "`test`" value in the `invalidName` property. If you enter this value, the app will display an error message specified in the `message` property.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        "UsrName": {
            ...,
            "validators": {
                /* Bind the custom validator to the attribute. */
                "MyValidator": {
                    "type": "usr.MyValidator",
                    "params": {
                        "invalidName": "test",
                        "message": "Invalid name"
                    }
                }
            }
        },
        ...
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

Complete source code of the page schema

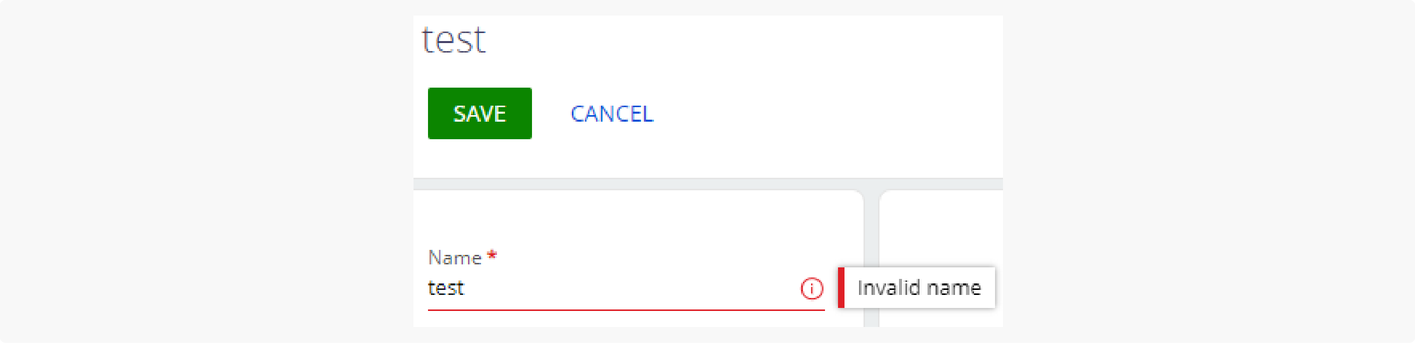
3. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

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

1. Open the `Validators` app page and click [*Run app*].
2. Click [*New*] on the `Validators` app toolbar.
3. Enter "`test`" in the [*Name*] field.
4. Click [*Save*] on the validator page toolbar.

As a result, the app will not save the `test` record and display an error notification in a pop-up box.

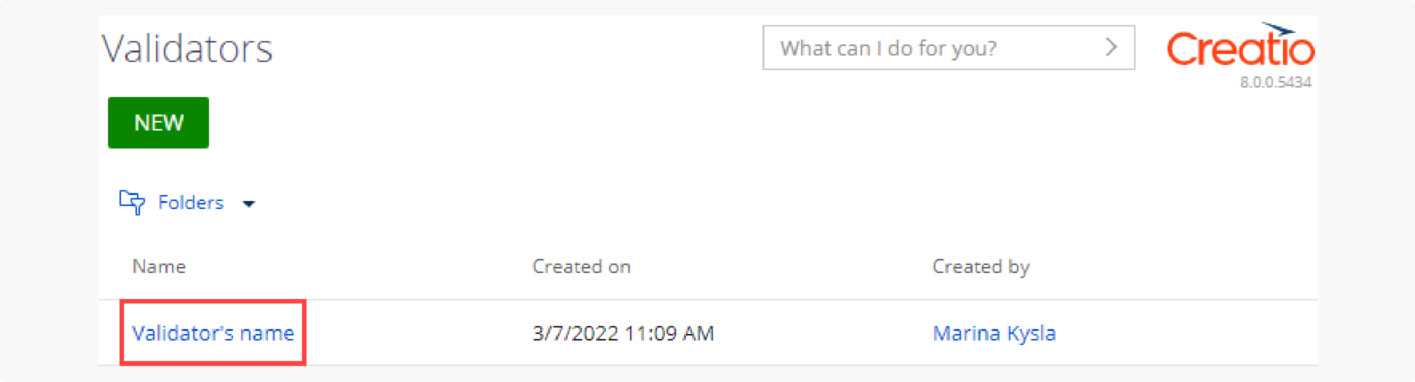


test

Name *
test

Invalid name

The app will save a record that has a different name, such as `Validator's name`, correctly. The record will be displayed in the `Validators` section list.



Validators

Name	Created on	Created by
Validator's name	3/7/2022 11:09 AM	Marina Kysla

Implement the field value conversion on a page

Medium

Example. Add a converter that converts the [*Name*] field value to uppercase to the record page of the custom [*Converters*] section. The [*Name*] field value must remain the same. Display the converted value in the [*Label*] type component.

1. Set up the page UI

1. Create a custom `Converters` app based on the [*Records & business processes*] template. To do this, follow the guide in user documentation: Create a custom app.
2. Open the [*Converters form page*] page in the working area of the `Converters` app page.

   The [*Converters form page*] page includes the [*Name*] field by default.
3. Add a [*Label*] type component to the working area of the Freedom UI Designer.
4. Click </> 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.

2. Set up the field value conversion

Configure the business logic in the Client Module Designer. For this example, set up the field value conversion. Convert the value of the [*Name*] field on the [*Validators form page*] page.

1. Implement a custom `usr.ToUpperCase` converter in the `converters` schema section.

   `converters` **schema section**

```
converters: /**SCHEMA_CONVERTERS*/{
    /* The custom converter. Converts the value to uppercase. */
    "usr.ToUpperCase": function(value) {
        return value?.toUpperCase() ?? '';
    }
}/**SCHEMA_CONVERTERS*/,
```

2. 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.ToUpperCase` converter to the `$UsrName` attribute.

   `viewConfigDiff` **schema section**

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
...,
{
    "operation": "insert",
    "name": "Label",
    "values": {
        ...,
        /* Bind the usr.ToUpperCase converter to the $UsrName attribute. */
```

```
            "caption": "$UsrName | usr.ToUpperCase",
            ...
        },
        ...
    }
    ]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

Complete source code of the page schema

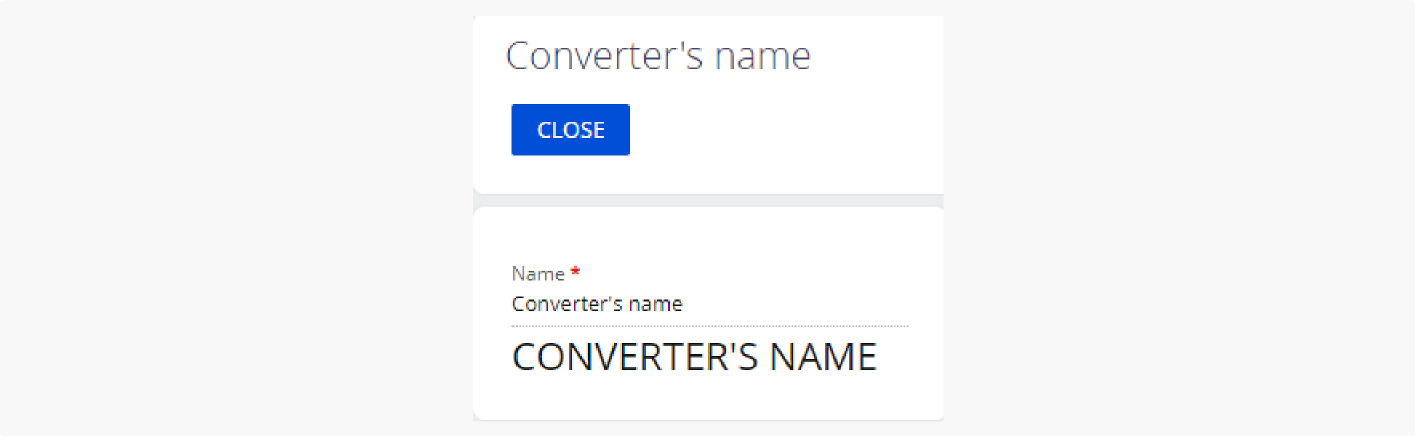
3. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

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

1. Open the `Converters` app page and click [*Run app*].
2. Click [*New*] on the `Converters` app toolbar.
3. Enter "`Converter's name`" in the [*Name*] field.

As a result, when you fill out the [*Name*] field on the converter page, Creatio will convert the field value to uppercase and display it in the [*Label*] type component. The [*Name*] field value will remain the same.



Name *
Converter's name
CONVERTER'S NAME

Display the value of a system variable

Beginner

Creatio 8 Atlas uses the `sdk.SysValuesService` service to manage system variables.

In Creatio 8 Atlas, accessing system variables is different from the previous versions. In this version, write the system variable name in lowercase without `_` delimiters and the `CURRENT` prefix (for example, `maintainer`, `primaryLanguage`, etc.).

To **display the value of a system variable on a page**:

1. Add a page inscription to display the values of system variables at step 1 of the Freedom UI page

customization procedure if needed.

2. Set up **how to display the value of a system variable on the page** at step 2 of the Freedom UI page customization procedure.

   a. Enable the `sdk.SysValuesService` system variable service. To do this, add the `@creatio-devkit/common` dependency to the AMD module.

      View an example that adds a dependency to the `UsrAppClientSchemaName` AMD module below.

      AMD module dependencies

   b. Add an attribute that stores data to the `viewModelConfig` schema section. Add the attribute similarly to the procedure for setting up the field display condition.

   c. Bind the `caption` property to the corresponding model attribute in the `viewConfigDiff` schema section. Property binding is similar to that described in the setup procedure for the field display condition. Instead of the `visible` property, use the `caption` property responsible for the text displayed in the element.

   d. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. The handler is executed when the `View` model is initialized. Depending on the value of the attribute (`true` or `false`), the handler executes different business logic.

      a. Instantiate the system value service from `@creatio-devkit/common`.

      b. Load system values.

      c. Calculate the value and write the calculation result to the corresponding attribute if needed.

      View an example of a `crt.HandlerViewModelInitRequest` query handler with `someVariable` calculation result written in the `SomeAttributeName` attribute below.

      `handlers` schema section

View a detailed example that configures how to display the value of a system variable in a separate article: Display the values of system variables on a page.

Display the values of system variables on a page

Medium

Example. Display the following on the record page of the custom [*System variables*] section:

- the name of the current user
- the time offset value in hours between the time zone of the current contact and universal time (UTC)

Retrieve the values from the corresponding system variables.

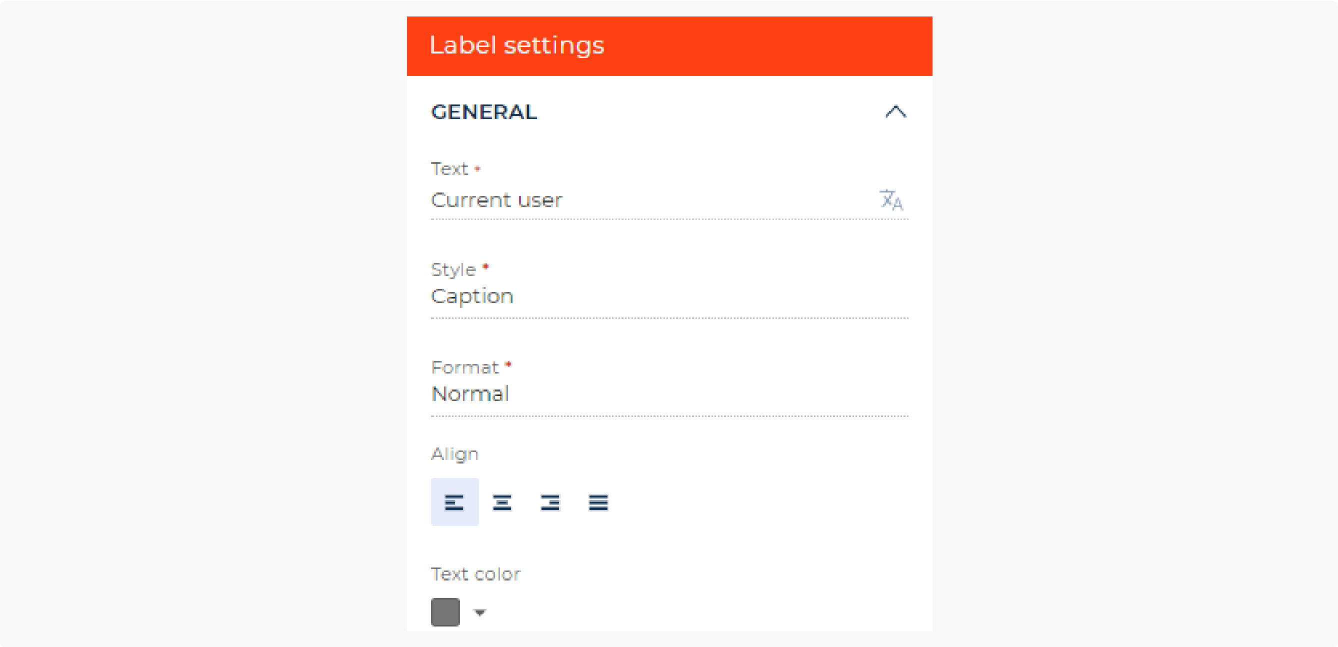
1. Set up the page UI

1. Create a custom `System variables` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: [Create a custom app](#).
2. Open the [*System variables form page*] page in the working area of the `System variables` app page.
3. Delete the [*Name*] field the [*Requests form page*] page includes by default.
4. Add the **label of the current contact**.

    a. Add a [*Label*] type component to the working area of the Freedom UI Designer.

    b. Click ☰ in the action panel of the Freedom UI Designer and fill out the **label properties** in the setup area.

    - Set [*Title*] to "`Current user`."
    - Select "`Caption`" in the [*Style*] property.
    - Select gray in the [*Text color*] property.



Label settings

GENERAL

Text *
Current user

Style *
Caption

Format *
Normal

Align

Text color

5. Add the following **labels** in a similar way:

    - the value of the current contact name from the system variable
    - the time offset from UTC
    - the time offset value from UTC from the system variable

    View the properties of the labels to add in the table below.

Label property values

Element	Property	Property value
The label that contains the value of the current contact name from the system variable	[*Title*]	"`Current user (value)`"
	[*Style*]	Select "`Body text`"
The label of the time offset from UTC	[*Title*]	"`Contact time offset`"
	[*Style*]	Select "`Caption`"
	[*Text color*]	Select the gray color
The label that contains the time offset value from UTC from the system variable	[*Title*]	"`Contact time offset (value)`"
	[*Style*]	Select "`Body text`"

6. Click </> 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.

2. Set up the retrieval of system variable values

Configure the business logic in the Client Module Designer. For this example, set up the retrieval of system variable values

1. Enable the `sdk.SysValuesService` system variable service. To do this, add `@creatio-devkit/common` to the AMD module as a dependency.

AMD module dependencies

```
/* Declare the AMD module. */
define("UsrAppSystemvariable_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_
    return {
        ...
    };
});
```

2. Add the following **attributes** to the `viewModelConfig` schema section:

- `CurrentUser`. Stores the name of the current contact.
- `ContactTimezone`. Stores the time offset in hours between the time zone of the current contact and UTC.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that stores the name of the current contact. */
        "CurrentUser": {},
        /* The attribute that stores the time offset in hours between the time zone of the cu
        "ContactTimezone": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

3. Change the `caption` **property value** in the `viewConfigDiff` schema section:

- `$CurrentUser` for the `CurrentUserValue` element
- `$ContactTimezone` for the `ContactTimeOffsetValue` element

The `caption` property handles the text contained in the element.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "CurrentUserValue",
        "values": {
            ...,
            /* Bind the CurrentUser attribute to the caption property. */
            "caption": "$CurrentUser",
            ...
        },
        ...
    },
    ...,
    {
        "operation": "insert",
        "name": "ContactTimeOffsetValue",
        "values": {
            ...,
            /* Bind the ContactTimezone attribute to the caption property. */
            "caption": "$ContactTimezone",
            ...
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

4. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. Execute the handler when Creatio initializes the `View` model.

   a. Create an instance of the system value service from `@creatio-devkit/common`.

   b. Upload the system values.

   c. Specify the name of the current user contact in the `CurrentUser` attribute.

   d. Convert the time zone offset value from minutes to hours and specify the converted value in the `ContactTimezone` attribute.

`handlers` schema section

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "crt.HandleViewModelInitRequest",
        /* The custom implementation of the system query handler. */
        handler: async (request, next) => {
            /* Create an instance of the system value service from @creatio-devkit/common. */
            const sysValuesService = new sdk.SysValuesService();
            /* Upload the system values. */
            const sysValues = await sysValuesService.loadSysValues();
            /* Specify the name of the current user contact in the CurrentUser attribute. */
            request.$context.CurrentUser = sysValues.userContact.displayValue;
            /* Convert the time zone offset value from minutes to hours and specify the conve
            const offset = sysValues.userTimezoneOffset;
            const offsetDisplayValue = (offset > 0 ? '+' : '') + (offset / 60) + 'h';
            request.$context.ContactTimezone = offsetDisplayValue;
            /* Call the next handler if it exists and return its result. */
            return next?.handle(request);
        }
    }
] /**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

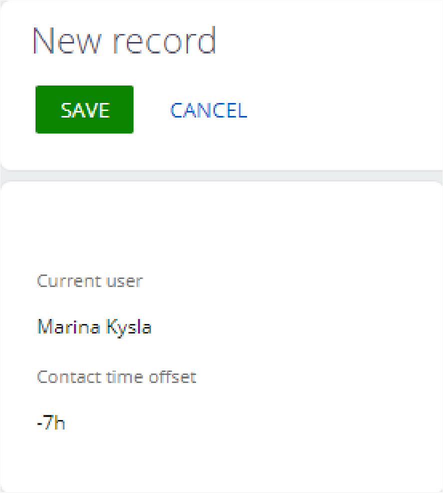
5. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

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

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

As a result, Creatio will display the name of the current user and the time offset value in hours between the time zone of the current contact and UTC on the record page of the custom [*System variables*] section. The values will be retrieved from the corresponding system variables.



New record

Current user

Marina Kysla

Contact time offset

-7h

Send a web service request and handle the response

Beginner

Creatio 8 Atlas uses the `sdk.HttpClientService` service to send web service requests.

To **send a web service request and handle the response on the page**:

1. Add a page inscription to display the handled result of a web service request at step 1 of the Freedom UI page customization procedure if needed.
2. Set up **how to send a web service request and handle the response on the page** at step 2 of the Freedom UI page customization procedure.
   a. Enable the `sdk.HttpClientService` service that sends HTTP requests. Enable the service similarly to the display procedure for the value of system variables.
   b. Add an attribute that stores data to the `viewModelConfig` schema section. Add the attribute similarly to the setup procedure for the field display condition.
   c. Bind the `caption` property to the corresponding model attribute in the `viewConfigDiff` schema section. Property binding is similar to that described in the setup procedure for the field display condition. Instead of the `visible` property, use the `caption` property responsible for the text displayed in the element.
   d. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. The handler is executed when the `View` model is initialized.

a. Instantiate the HTTP client from `@creatio-devkit/common`.

b. Specify the URL to get the required information. If a web service request is sent using a non-absolute path (without `https://` or `http://` prefixes), this is a request to an internal Creatio web service. In that case, Creatio automatically adds the address of the current app to the link.

c. Send a GET request.

d. Retrieve the required values from the response and write them to the corresponding attributes.

View an example of the `crt.HandlerViewModelInitRequest` request handler that sends a request to the `https://SomeUrlValue` web service, receives the `someValue` parameter from the response body, and writes the parameter to the `SomeAttributeName` attribute, below.

`handlers` schema section

View a detailed example that sets up a web service request and handles the response in a separate article: Send a request to an external web service and handle its result on a page.

Send a request to an external web service and handle its result on a page

Medium

Example. Call the `coindesk.com` external web service from the record page of the [*Http Client Service*] custom section. Display the current BTC to the USD exchange rate.

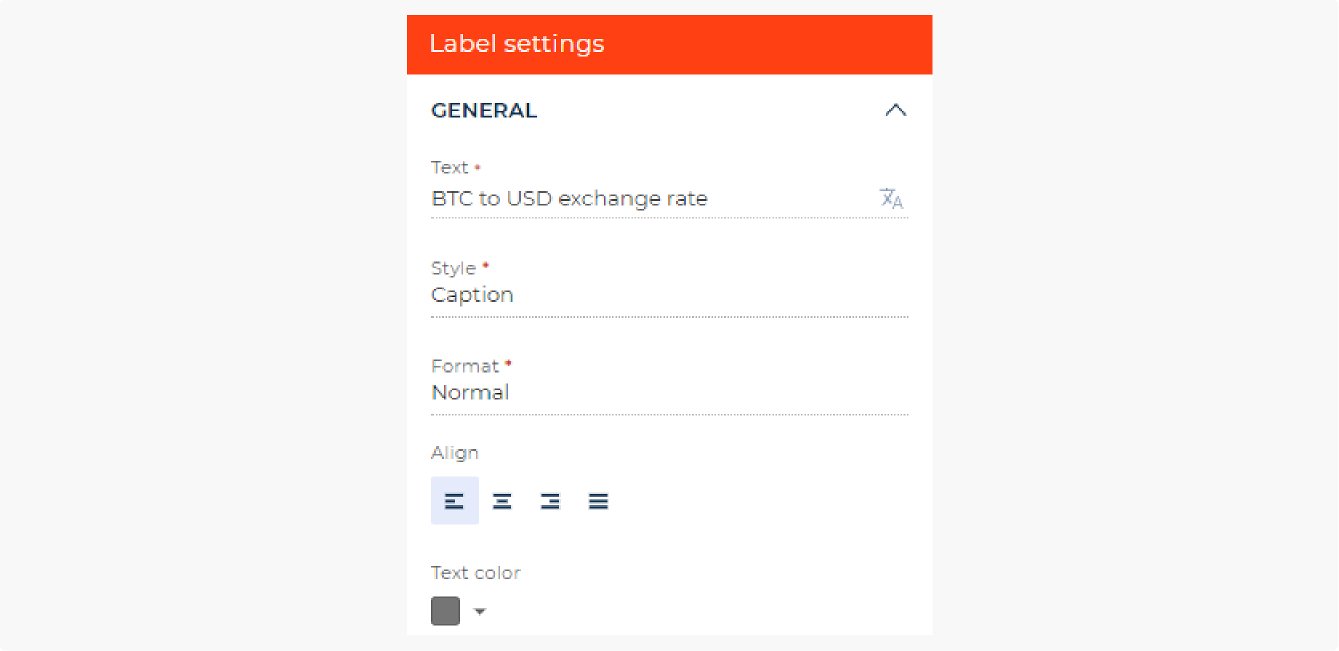
1. Set up the page UI

1. Create a custom `Http Client Service` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
2. Open the [*Http Client Service form page*] page in the working area of the `Http Client Service` app page.
3. Delete the [*Name*] field the [*Http Client Service form page*] page includes by default.
4. Add a **label of the BTC to the USD exchange rate**.

   a. Add a [*Label*] type component to the working area of the Freedom UI Designer.

   b. Click in the action panel of the Freedom UI Designer and fill out the **label properties** in the setup area.

   - Set [*Title*] to "`BTC to USD exchange rate`."
   - Select "`Caption`" in the [*Style*] property.
   - Select gray in the [*Text color*] property.



Label settings

GENERAL

Text *
BTC to USD exchange rate

Style *
Caption

Format *
Normal

Align

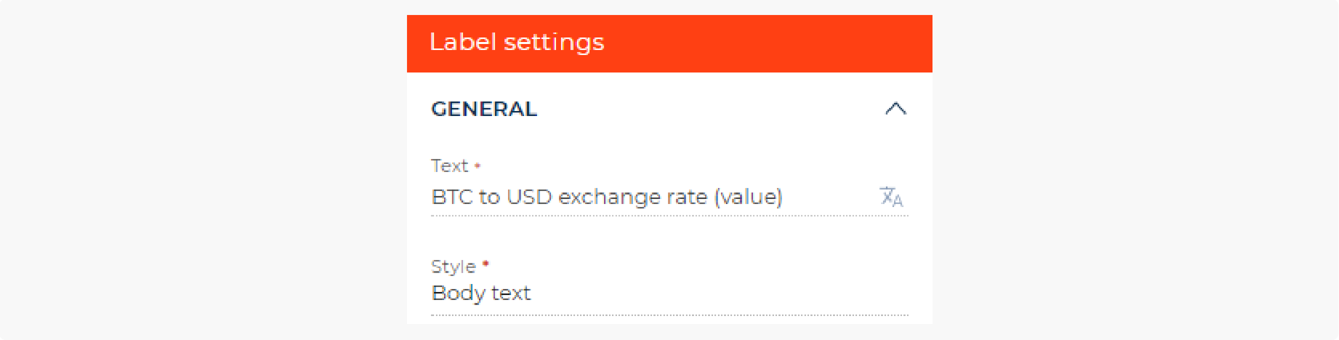
Text color

5. Add a **label that contains the value of the BTC to the USD exchange rate**.

   a. Add a [*Label*] type component to the working area of the Freedom UI Designer.

   b. Click in the action panel of the Freedom UI Designer and fill out the **label properties** in the setup area.

      - Set [*Title*] to "`BTC to USD exchange rate (value)`."
      - Select "`Body text`" in the [*Style*] property.



Label settings

GENERAL

Text *
BTC to USD exchange rate (value)

Style *
Body text

6. Click 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.

2. Send the web service request and handle its results

Configure the business logic in the Client Module Designer. For this example, send the web service request and handle its results.

1. Enable the `sdk.HttpClientService` service that sends HTTP requests. To do this, add `@creatio-devkit/common` to the AMD module as a dependency.

AMD module dependencies

```
/* Declare the AMD module. */
define("UsrAppHttpClientServ_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_
        ...
    };
});
```

2. Add the `BtcToUsd` attribute that stores data of the bitcoin to dollar exchange rate to the `viewModelConfig` schema section.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that stores the bitcoin to dollar exchange rate. */
        "BtcToUsd": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

3. Bind the `caption` property of the `BtcToUsdExchangeRateValue` element to the `$BtcToUsd` model attribute in the `viewConfigDiff` schema section. The `caption` property handles the text contained in the element.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "BtcToUsdExchangeRateValue",
        "values": {
            ...,
            /* Bind the BtcToUsd attribute to the caption property. */
            "caption": "$BtcToUsd",
            ...
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

4. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. Execute the handler when Creatio initializes the `View` model.

   a. Create an instance of the HTTP client from `@creatio-devkit/common`.

b. Specify the URL to retrieve the current rate. Use the `coindesk.com` external web service.

c. Send a GET request.

d. Retrieve the rate from the response and specify the rate in the `BtcToUsd` attribute.

`handlers` **schema section**

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "crt.HandlerViewModelInitRequest",
        /* The custom implementation of the system query handler. */
        handler: async (request, next) => {
            /* Create an instance of the HTTP client from @creatio-devkit/common. */
            const httpClientService = new sdk.HttpClientService();
            /* Specify the URL to retrieve the current rate. Use the coindesk.com external se
            const endpoint = "https://api.coindesk.com/v1/bpi/currentprice/USD.json";
            /* Send a GET request. The HTTP client converts the response body from JSON to a
            const response = await httpClientService.get(endpoint);
            /* Retrieve the rate from the response and specify the rate in the BtcToUsd attri
            request.$context.BtcToUsd = response.body.bpi.USD.rate;
            /* Call the next handler if it exists and return its result. */
            return next?.handle(request);
        },
    },
] /**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

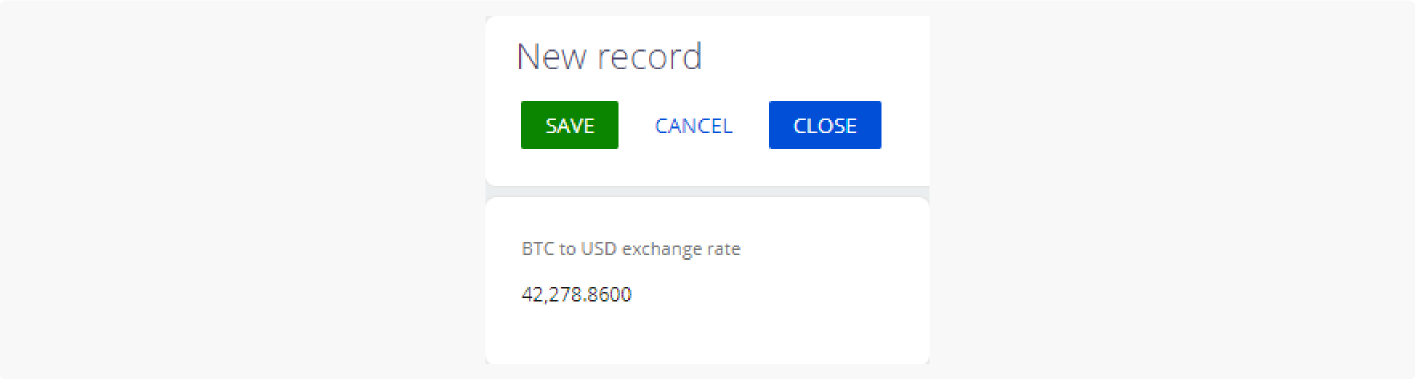
5. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

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

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

As a result, Creatio will display the current BTC to the USD exchange rate on the record page of the [*Http Client Service*] custom section. The value will be retrieved from the `coindesk.com` external web service.



BTC to USD exchange rate

42,278.8600

Hide functionality on a page

Beginner

Creatio 8 Atlas lets you execute the following **actions** that hide functionality on a page:

- Hide functionality during development.
- Hide functionality due to insufficient access permissions.

Hide functionality during development

Creatio 8 Atlas uses the `sdk.FeatureService` service to check functionality status.

To **hide functionality during development on a page** :

1. Add a page component that contains the functionality during development at step 1 of the [Freedom UI page customization procedure]() if needed.
2. Set up **how to hide the functionality on the page** at step 2 of the [Freedom UI page customization procedure]().

    a. Enable the `sdk.FeatureService` service that checks the functionality status. Enable the service similarly to the [display procedure for the value of system variables]().

    b. Add an attribute that stores data to the `viewModelConfig` schema section. Add the attribute similarly to the [setup procedure for the field display condition]().

    c. Bind the `visible` property to the corresponding model attribute in the `viewConfigDiff` schema section. Property binding is similar to that described in the [setup procedure for the field display condition]().

    d. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. The handler is executed when the `View` model is initialized.

        a. Instantiate the service that checks the functionality status from `@creatio/sdk`.

        b. Get the status of the functionality by its code and write it to the corresponding attribute.

    View an example of a `crt.HandlerViewModelInitRequest` query handler that receives a feature status with the `SomeFeatureCode` code and writes it to the `SomeAttributeName` attribute below.

    `handlers` schema section

View a detailed example that hides functionality during development in a separate article: Hide a feature at the development stage on a page.

Hide functionality due to insufficient access permissions

Creatio 8 Atlas uses the `sdk.RightsService` service to check access permissions.

To **hide functionality due to insufficient access permissions**:

1. Add a page component with the functionality for which access permissions are configured at step 1 of the Freedom UI page customization procedure if needed.
2. Set up **how to hide functionality on the page due to insufficient access permissions** at step 2 of the Freedom UI page customization procedure.
    a. Enable the `sdk.RightsService` service that checks access permissions. Enable the service similarly to the display procedure for the value of system variables.
    b. Add an attribute that stores data to the `viewModelConfig` schema section. Add the attribute similarly to the setup procedure for the field display condition.
    c. Bind the `visible` property to the corresponding model attribute in the `viewConfigDiff` schema section. Property binding is similar to that described in the setup procedure for the field display condition.
    d. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. The handler is executed when the `View` model is initialized.
        a. Instantiate the service that checks access permissions from `@creatio/sdk`.
        b. Get information about the user’s permissions to perform the corresponding action.
        c. Write the result of the checkup to the corresponding attribute.

        View an example of the `crt.HandlerViewModelInitRequest` request handler that checks for permissions to perform a system operation with the `SomeOperationCode` code and writes the result to the `SomeAttributeName` attribute below.

        `handlers` schema section

View a detailed example that hides functionality due to insufficient access permissions in a separate article: Hide the feature on a page due to insufficient access permissions.

Hide a feature at the development stage on a page

Medium

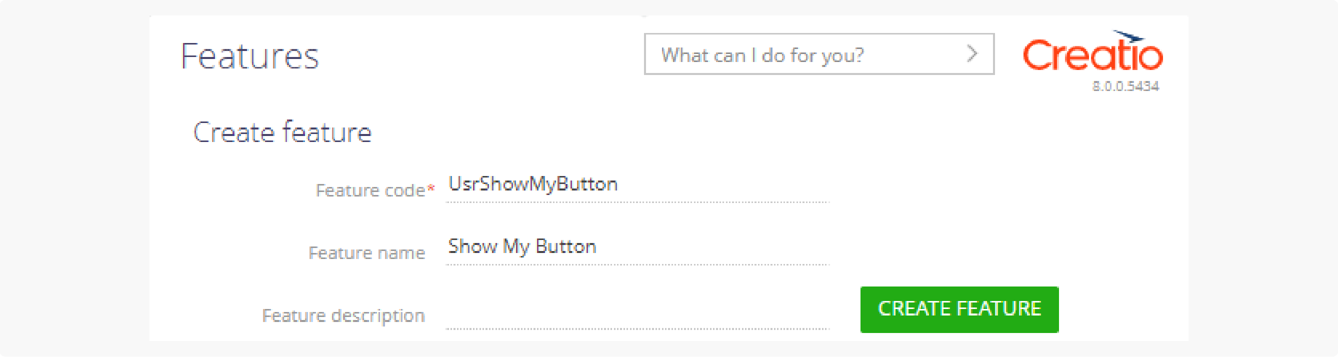
Example. Hide a custom [*Feature*] button on a record page of a custom [*Feature Service*] section. The button contains the feature at the development stage.

1. Set up the page UI

1. Add the **developed feature to hide**.

   a. Open the [*Feature*] page and fill out the **feature properties**:

   - Set [*Feature code*] to "`UsrShowMyButton`."
   - Set [*Feature name*] to "`Show My Button`."



Features

What can I do for you?

Creatio 8.0.0.5434

Create feature

Feature code* UsrShowMyButton

Feature name Show My Button

Feature description

CREATE FEATURE

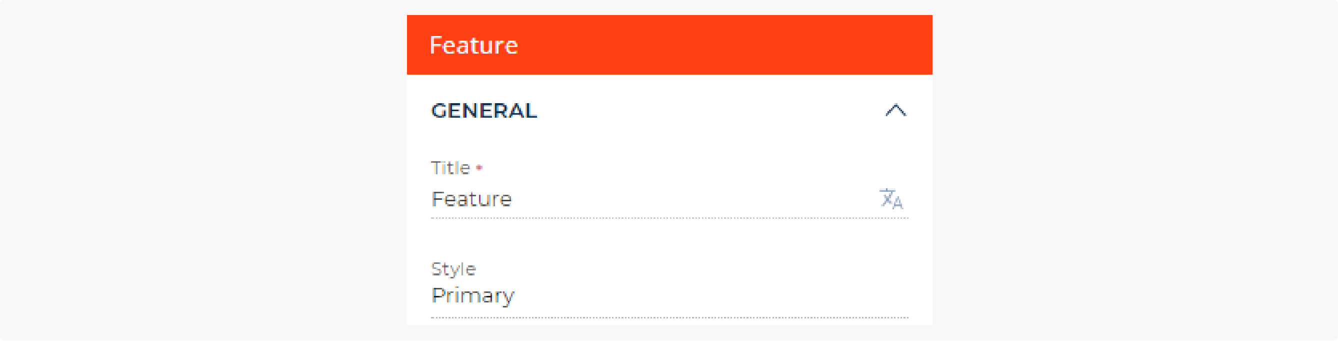
   d. Click [*Create feature*].

2. Create a custom `Feature Service` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
3. Open the [*Feature service form page*] page in the working area of the `Feature Service` app page.
4. Delete the [*Name*] field the [*Feature Service form page*] page includes by default.
5. Add a **button that contains the feature at the development stage**.

   a. Add a [*Button*] type component to the toolbar of the Freedom UI Designer.

   b. Click in the action panel of the Freedom UI Designer and fill out the **button properties** in the setup area.

   - Set [*Title*] to "`Feature`."
   - Select "`Primary`" in the [*Style*] property.



Feature

GENERAL

Title * Feature

Style Primary

6. Click </> 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.

2. Hide the feature at the development stage

Configure the business logic in the Client Module Designer. For this example, hide the feature at the development stage.

1. Enable the `sdk.FeatureService` service that checks the feature status. To do this, add `@creatio-devkit/common` to the AMD module as a dependency.

   AMD module dependencies

   ```
   /* Declare the AMD module. */
   define("UsrAppFeatureService_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_
           ...
       };
   });
   ```

2. Add the `ShowMyButton` attribute that stores feature status data to the `viewModelConfig` schema section.

   `viewModelConfig` schema section

   ```
   viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
       "attributes": {
           ...,
           /* The attribute that stores the feature status. */
           "ShowMyButton": {}
       }
   }/**SCHEMA_VIEW_MODEL_CONFIG*/,
   ```

3. Bind the `visible` property of the `FeatureButton` element to the `ShowMyButton` model attribute in the `viewConfigDiff` schema section.

   `viewConfigDiff` schema section

   ```
   viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
       {
           "operation": "insert",
           "name": "FeatureButton",
           "values": {
               ...,
               /* The property that flags the field as visible. Bound to the ShowMyButton attrib
               "visible": "$ShowMyButton"
           },
           ...
       }
   ]/**SCHEMA_VIEW_CONFIG_DIFF*/,
   ```

4. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. Execute the handler when Creatio initializes the `View` model.

   a. Create an instance of the service that checks the feature status from `@creatio-devkit/common`.

   b. Retrieve the status of the feature that has the `UsrShowMyButton` code and specify the status in the `ShowMyButton` attribute.

   `handlers` schema section

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "crt.HandleViewModelInitRequest",
        /* The custom implementation of the system query handler. */
        handler: async (request, next) => {
            /* Create an instance of the service that checks the feature status from @creatio
            const featureService = new sdk.FeatureService();
            /* Retrieve the UsrShowMyButton feature status and specify it in the ShowMyButton
            request.$context.ShowMyButton = await featureService.getFeatureState('UsrShowMyBu
            /* Call the next handler if it exists and return its result. */
            return next?.handle(request);
        }
    }
] /**SCHEMA_HANDLERS*/,
```

   Complete source code of the page schema

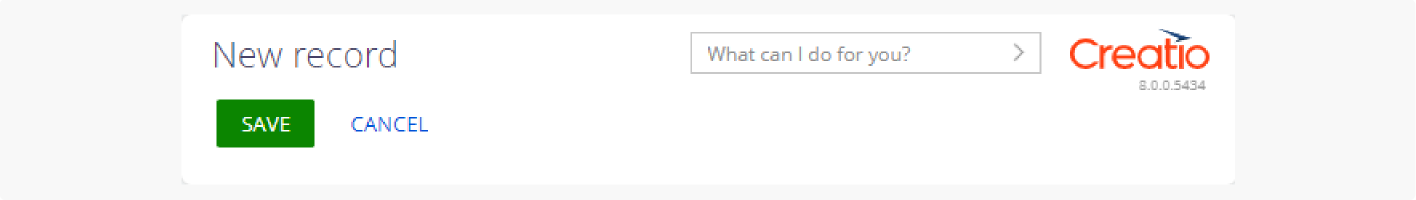
5. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

To **view the outcome of the example for the feature at the development stage**:

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

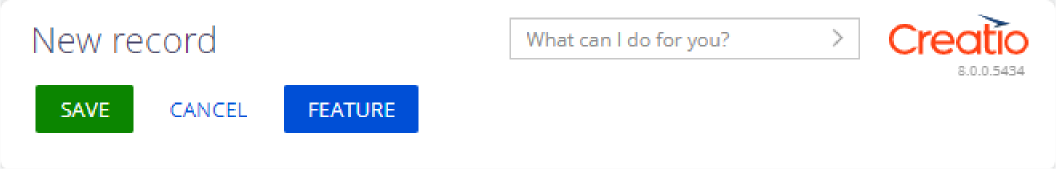
As a result, Creatio will hide the [*Feature*] button that contains the feature at the development stage on the `Feature Service` app page.



To **view the outcome of the example for the fully developed feature**:

1. Enable the feature that has `UsrShowMyButton` code.
    a. Open the [*Feature*] page.
    b. Enable the `Show My Button` feature in the page list.
    c. Click [*Save changes*] and refresh the page.
2. Refresh the `Feature Service` app page.
3. Click [*New*] on the `Feature Service` app toolbar.

As a result, Creatio will display the [*Feature*] button that contains the fully developed feature on the `Feature Service` app page.



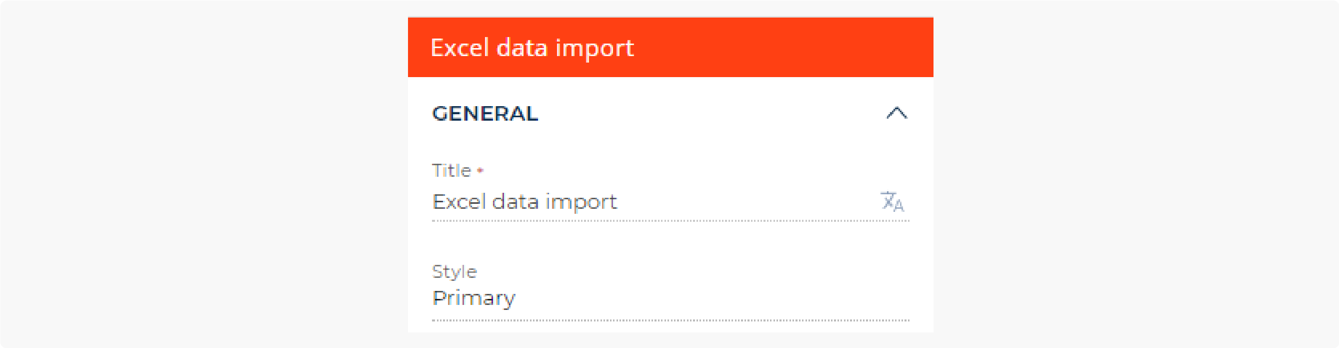
Hide the feature on a page due to insufficient access permissions

Medium

Example. Hide the [*Excel data import*] button on the record page of the custom [*Rights Service*] section if the user lacks permission to import Excel data.

1. Set up the page UI

1. Create a custom `Rights Service` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
2. Open the [*Rights Service form page*] page in the working area of the `Rights Service` app page.
3. Delete the [*Name*] field the [*Rights Service form page*] page includes by default.
4. Add a **button that starts Excel data import**.
    a. Add a [*Button*] type component to the toolbar of the Freedom UI Designer.
    b. Click ☰ in the action panel of the Freedom UI Designer and fill out the **button properties** in the setup area.
        - Set [*Title*] to "`Excel data import`."
        - Select "`Primary`" in the [*Style*] property.



Excel data import

GENERAL

Title *
Excel data import

Style
Primary

5. Click </> 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.

2. Hide the feature if the user lacks permission to access it

Configure the business logic in the Client Module Designer. For this example, hide the feature if the user lacks permission to access it

1. Enable the `sdk.RightsService` that checks access permissions. To do this, add `@creatio-devkit/common` to the AMD module as a dependency.

AMD module dependencies

```
/* Declare the AMD module. */
define("UsrAppRightsService_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_D
        ...
    };
});
```

2. Add the `CanImportFromExcel` attribute that stores the user's access permission data to the `viewModelConfig` schema section.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that stores the user's access permission data. */
        "CanImportFromExcel": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

3. Bind the `visible` property of the `ExcelDataImportButton` element to the `CanImportFromExcel` model attribute in the `viewConfigDiff` schema section. The `visible` property flags the button as visible.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "insert",
        "name": "ExcelDataImportButton",
        "values": {
            ...,
            /* The property that flags the button as visible. Bound to the CanImportFromExcel
            "visible": "$CanImportFromExcel"
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

4. Add a custom implementation of the `crt.HandlerViewModelInitRequest` system query handler to the `handlers` schema section. Execute the handler when Creatio initializes the `View` model.

   a. Create an instance of the service that checks access permissions from `@creatio-devkit/common`.

   b. Retrieve data about the user's access permission to the `CanImportFromExcel` system operation.

   c. Specify the data in the `CanImportFromExcel` attribute.

`handlers` schema section

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "crt.HandleViewModelInitRequest",
        /* The custom implementation of the system query handler. */
        handler: async (request, next) => {
            /* Create an instance of the service that checks access permissions from @creatio
            const rightService = new sdk.RightsService();
            /* Retrieve data about the user's access permission to the CanImportFromExcel sys
            const canImportFromExcel = await rightService.getCanExecuteOperation('CanImportFr
            /* Specify the data in the CanImportFromExcel attribute. */
            request.$context.CanImportFromExcel = canImportFromExcel;
            /* Call the next handler if it exists and return its result. */
            return next?.handle(request);
        },
    }
] /**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

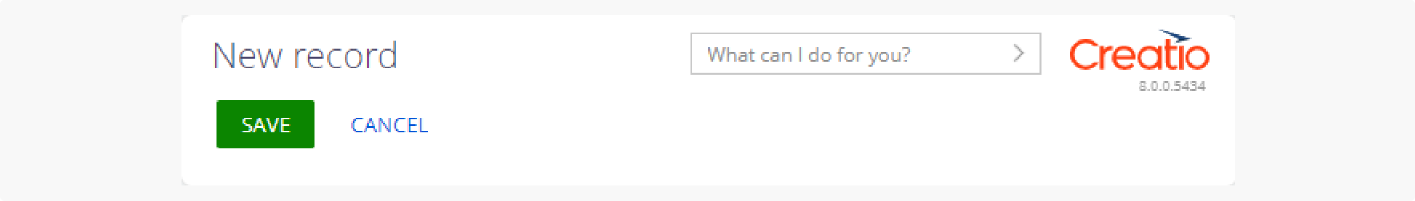
5. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

To **view the outcome of the example without the access permission**:

1. Log in to the app as a user who lacks the permission to import data from Excel. For example, create a new user or revoke the permission from an existing user. To add a user, follow the guide in the user documentation: Add users. To set up access permissions, follow the guide in the user documentation: System operation permissions. The [*Excel import*] (`CanImportFromExcel` code) system operation manages Excel data import.
2. Open the `Rights Service` app page and click [*Run app*].
3. Click [*New*] on the `Rights Service` app toolbar.

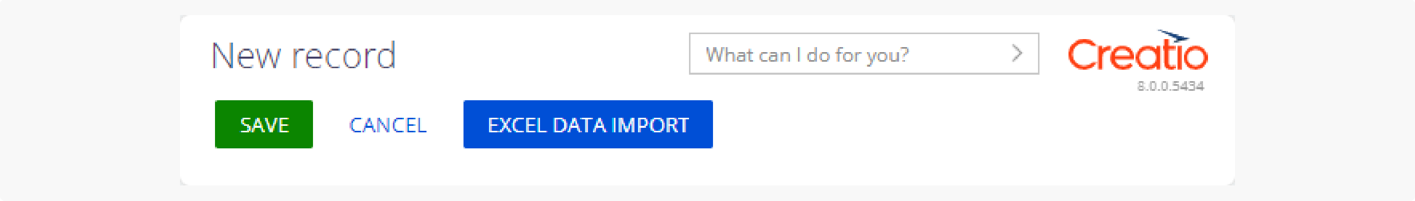
As a result, Creatio will hide the [*Excel data import*] button that starts Excel data import on the `Rights Service` app page.



To **view the outcome of the example with the access permission**:

1. Refresh the `Rights Service` app page.
2. Click [*New*] on the `Rights Service` app toolbar.

As a result, Creatio will display the [*Excel data import*] button that starts Excel data import on the `Rights Service` app page.



Open a page from a custom handler

Beginner

Creatio 8 Atlas uses the `sdk.HandlerChainService` service to open pages. Creatio 7.X and Creatio 8 Atlas use the same method to open record pages. You can pass the needed default column values when Creatio adds a record.

Creatio 8 Atlas provides the following **actions** to open pages from a custom handler:

- Open a record page from a custom handler.
- Open a Freedom UI page from a custom handler.

Open a record page from a custom handler

1. Add a page button that opens the record page on click at step 1 of the Freedom UI page customization procedure if needed.
2. Set up **how to open the record page from a custom handler** at step 2 of the Freedom UI page customization procedure.

   a. Enable the `sdk.HandlerChainService` service that opens pages. Enable the service similarly to the display procedure for the value of system variables.

   b. Bind the `clicked` property to the corresponding query in the `viewConfigDiff` schema section. Describe the business logic that opens the page in the `handlers` schema section. The `clicked` property is responsible for the action performed on button click.

   View an example that binds the `clicked` property to the `usr.SomeRequest` custom query below.

   `viewConfigDiff` schema section

   c. Add the implementation of a custom query to the `handlers` schema section.

   To **open a page**:

   a. Get an instance of the `sdk.HandlerChainService` singleton service that opens pages.

   b. Send a `crt.UpdateRecordRequest` system query that opens the page by the specified ID.

   View an example of the `usr.SomeRequest` query handler that sends the `crt.UpdateRecordRequest` system query below. The `crt.UpdateRecordRequest` query opens the page of the `SomeSchemaName` record with the `SomeRecordId` ID.

   `handlers` schema section

   To **open the page and populate the fields with the specified values**:

   a. Get an instance of the `sdk.HandlerChainService` singleton service that opens pages.

   b. Send the `crt.CreateRecordRequest` system query that creates a page with fields populated with the specified values.

   View an example of the `usr.SomeRequest` query handler that sends the `crt.CreateRecordRequest` system query below. The `crt.CreateRecordRequest` query opens the `SomeSchemaName` record page and populates the [*SomeField*] field with the "`SomeRecordId`" value.

   `handlers` schema section

View a detailed example that opens a record page in a separate article: Open a record page from a custom handler.

Open a Freedom UI page from a custom handler

1. Take steps 1-2 from the procedure to open the record page from a custom handler.

2. Set up **how to open a Freedom UI page from a custom handler** at step 2 of the procedure for opening the record page from a custom handler. To do this, add the implementation of a custom query to the `handlers` schema section.

   a. Get an instance of the `sdk.HandlerChainService` singleton service that opens pages.

   b. Send a `crt.OpenPageRequest` system query that opens the Freedom UI page with the specified name.

   View an example `usr.SomeRequest` query handler that sends the `crt.OpenPageRequest` system query below. The `crt.OpenPageRequest` query opens the `SomePageName` page.

   `handlers` schema section

View a detailed example that opens a Freedom UI page in a separate article: Open a Freedom UI page from a custom handler.

Open a record page from a custom handler

Medium

Example. Add the following buttons to the record page of the custom [*Handler Chain Service*] section:

- [*Edit contact*]. Must open the page of the contact that has the specified ID.
- [*Create request*]. Must open the page of a new request in the custom [*Requests*] section. The [*Name*] field must be populated with the "`New request`" value.

Records open similarly in Creatio versions 7.X and 8.X. When Creatio adds a new record, you can pass the needed default column values.

1. Set up the UI of the pages

1. Set up the UI of the custom [*Requests*] **section page**. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.

   The [*Requests form page*] page includes the [*Name*] field by default.

2. Set up the UI of the custom [*Handler Chain Service*] **section page**.

   a. Click ← on the `Requests` app page.

   b. Create a custom `Handler Chain Service` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.

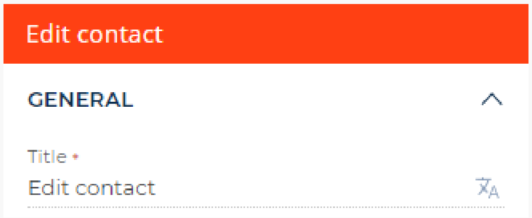
   c. Open the [*Handler Chain Service form page*] page in the working area of the `Handler Chain Service` app page.

   d. Delete the [*Name*] field the [*Handler Chain Service form page*] page includes by default.

   e. Add a **button that opens the page of the contact that has the specified ID**.

a. Add a [*Button*] type component to the toolbar of the Freedom UI Designer.

b. Click in the action panel of the Freedom UI Designer and set the [*Title*] button property in the setup area to "`Edit contact`."



Edit contact

GENERAL

Title *

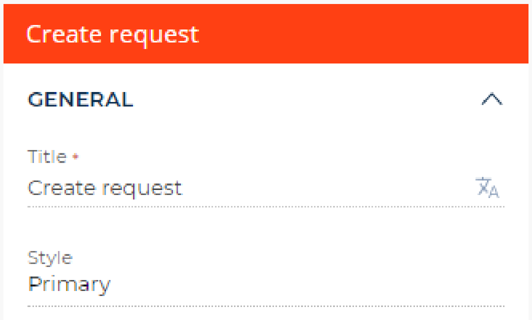
Edit contact

f. Add a **button that opens the page of a new request** in the custom [*Requests*] section.

a. Add a [*Button*] type component to the toolbar of the Freedom UI Designer.

b. Click in the action panel of the Freedom UI Designer and fill out the **button properties** in the setup area.

- Set [*Title*] to "`Create request`."
- Select "`Primary`" in the [*Style*] property.



Create request

GENERAL

Title *

Create request

Style

Primary

g. Click </> 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.

2. Set up the way record pages open

Configure the business logic in the Client Module Designer. For this example, set up the way record pages open.

1. Enable the `sdk.HandlerChainService` service that opens pages. To do this, add `@creatio-devkit/common` to the AMD module as a dependency.

AMD module dependencies

```
/* Declare the AMD module. */
define("UsrAppHandlerChainSe_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_
        ...
    };
```

```
});
```

2. Change the `clicked` **property value** in the `viewConfigDiff` schema section to the following:

- `usr.EditContactRequest` for the `EditContactButton` element
- `usr.CreateUsrRequestRequest` for the `CreateRequestButton` element

The `clicked` property handles the action executed on button click.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "insert",
        "name": "EditContactButton",
        "values": {
            ...,
            "clicked": {
                /* Bind the sending of the custom usr.EditContactRequest query to the clicked
                "request": "usr.EditContactRequest"
            }
        },
        ...
    },
    {
        "operation": "insert",
        "name": "CreateRequestButton",
        "values": {
            ...,
            "clicked": {
                /* Bind the sending of the custom usr.CreateUsrRequestRequest query to the cl
                "request": "usr.CreateUsrRequestRequest"
            }
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

3. **Implement custom query handlers** in the `handlers` schema section:

- `usr.EditContactRequest`
  - Retrieve the instance of the `sdk.HandlerChainService` singleton service that opens pages.
  - Send the `crt.UpdateRecordRequest` system query that opens the page of the contact that has the specified ID. You can view the ID of the contact whose page to open in the browser address bar. For this example, open the page of the Alexander Wilson contact whose ID is "`98dae6f4-70ae-4f4b-9db5-e4fcb659ef19`."

- `usr.CreateUsrRequestRequest`
  - Retrieve the instance of the `sdk.HandlerChainService` singleton service that opens pages.
  - Send the `crt.CreateRecordRequest` system query that opens the page of a new request. Populate the [*Name*] field with the "`New request`" value.

`handlers` schema section

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "usr.EditContactRequest",
        /* Implement the custom query handler. */
        handler: async (request, next) => {
            /* Retrieve the instance of the singleton service that opens pages. */
            const handlerChain = sdk.HandlerChainService.instance;
            /* Send the crt.UpdateRecordRequest system query that opens the page of the conta
            await handlerChain.process({
                type: 'crt.UpdateRecordRequest',
                entityName: 'Contact',
                recordId: '98dae6f4-70ae-4f4b-9db5-e4fcb659ef19',
                $context: request.$context
            });
            /* Call the next handler if it exists and return its result. */
            return next?.handle(request);
        }
    },
    {
        request: "usr.CreateUsrRequestRequest",
        /* Implement the custom query handler. */
        handler: async (request, next) => {
            /* Retrieve the instance of the singleton service that opens pages. */
            const handlerChain = sdk.HandlerChainService.instance;
            /* Send the crt.CreateRecordRequest system query that opens the page of a new req
            await handlerChain.process({
                type: 'crt.CreateRecordRequest',
                entityName: 'UsrAppRequests',
                        defaultValues: [{
                            attributeName: 'UsrName',
                            value: 'New request'
                }],
                $context: request.$context
            });
            /* Call the next handler if it exists and return its result. */
            return next?.handle(request);
        }
    }
] /**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

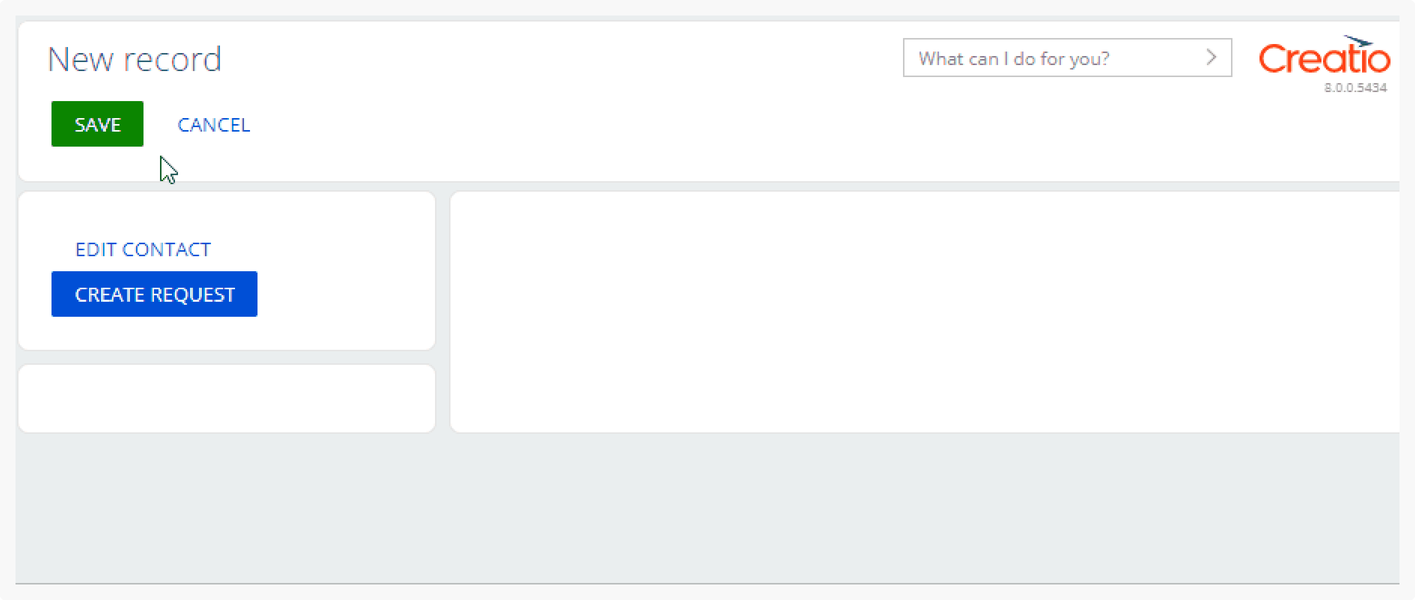
4. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

To **view the outcome of the example that opens the contact page**:

1. Open the `Handler Chain Service` app page and click [*Run app*].
2. Click [*New*] on the `Handler Chain Service` app toolbar.
3. Click [*Edit contact*] on the record page of the custom `Handler Chain Service` section.

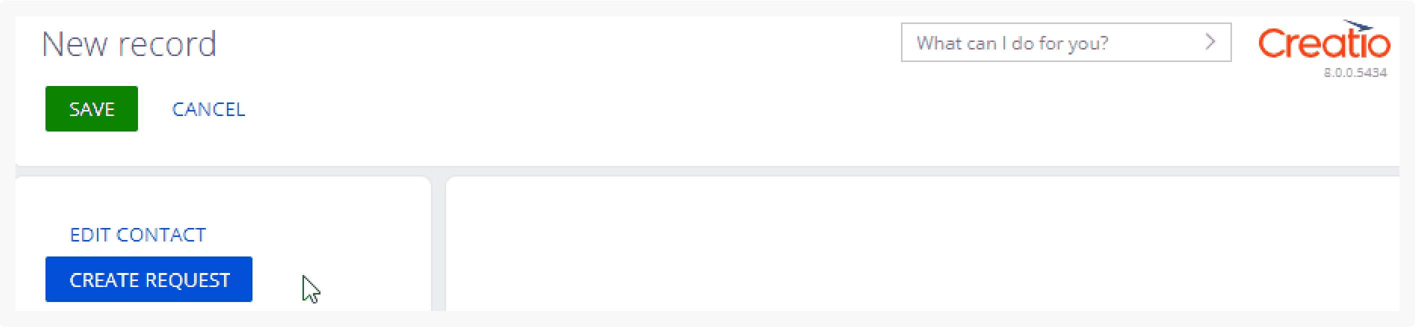
As a result, Creatio will open the page of the Alexander Wilson contact whose ID is "`98dae6f4-70ae-4f4b-9db5-e4fcb659ef19`."



To **view the outcome of the example that opens the request page and populates the field**:

1. Refresh the `Handler Chain Service` app page.
2. Click [*New*] on the `Handler Chain Service` app toolbar.
3. Click [*Create request*] on the record page of the custom `Handler Chain Service` section.

As a result, Creatio will open the page of a new request in the custom [*Requests*] section. The [*Name*] field will be populated with the "`New request`" value.



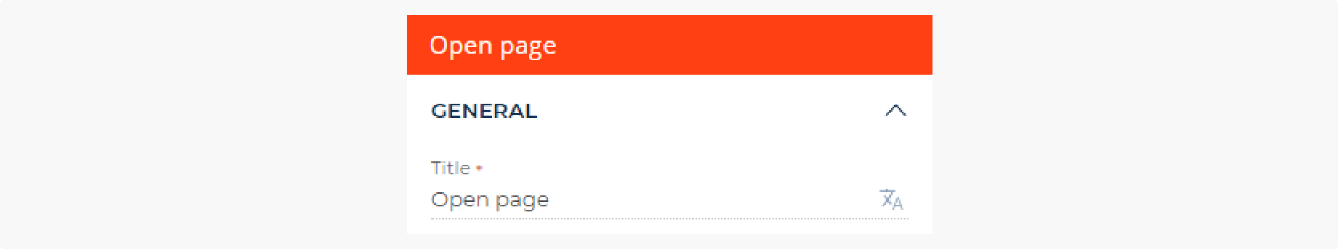
Open a Freedom UI page from a custom handler

Medium

Example. Add an [*Open page*] button to the record page of the custom [*Handler Chain Service*] section. The button must open the `StudioHomePage` Freedom UI page.

1. Set up the page UI

1. Create a custom `Handler Chain Service` app based on the [*Records & business processes*] template. To do this, follow the guide in the user documentation: Create a custom app.
2. Open the [*Handler Chain Service form page*] page in the working area of the `Handler Chain Service` app page.
3. Add a **button** that opens the `StudioHomePage` page.
   a. Add a [*Button*] type component to the toolbar of the Freedom UI Designer.
   b. Click ☰ in the action panel of the Freedom UI Designer and set the [*Title*] button property in the setup area to "`Open page`."



Title *
Open page

4. Click </> 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.

2. Set up the way the Freedom UI page opens

Configure the business logic in the Client Module Designer. For this example, set up the way the Freedom UI page opens.

1. Enable the `sdk.HandlerChainService` service that opens pages. To do this, add `@creatio-devkit/common` to the AMD module as a dependency.

AMD module dependencies

```
/* Declare the AMD module. */
define("UsrAppHandlerChainSe_FormPage", /**SCHEMA_DEPS*/["@creatio-devkit/common"] /**SCHEMA_
        ...
    };
});
```

2. Change the `clicked` property value for the `OpenPageButton` element in the `viewConfigDiff` schema section to `usr.OpenUsrTestPageRequest`. The `clicked` property handles the action executed on button click.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "insert",
        "name": "OpenPageButton",
        "values": {
            ...,
            "clicked": {
                /* Bind the sending of the custom usr.OpenUsrTestPageRequest query to the cli
                "request": "usr.OpenUsrTestPageRequest"
            }
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

3. Implement the `usr.OpenUsrTestPageRequest` custom query handler in the `handlers` schema section.
    a. Retrieve the instance of the `sdk.HandlerChainService` singleton service that opens pages.
    b. Send the `crt.OpenPageRequest` system query that opens the `StudioHomePage` page.

`handlers` schema section

```
handlers: /**SCHEMA_HANDLERS*/[
    {
        request: "usr.OpenUsrTestPageRequest",
        /* Implement the custom query handler. */
        handler: async (request, next) => {
            /* Retrieve the instance of the singleton service that opens pages. */
```

```
                const handlerChain = sdk.HandlerChainService.instance;
                /* Send the crt.OpenPageRequest system query that opens the Freedom UI page. */
                await handlerChain.process({
                    type: 'crt.OpenPageRequest',
                    schemaName: 'StudioHomePage',
                    $context: request.$context
                });
                /* Call the next handler if it exists and return its result. */
                return next?.handle(request);
            }
        }
    ] /**SCHEMA_HANDLERS*/,
```

Complete source code of the page schema

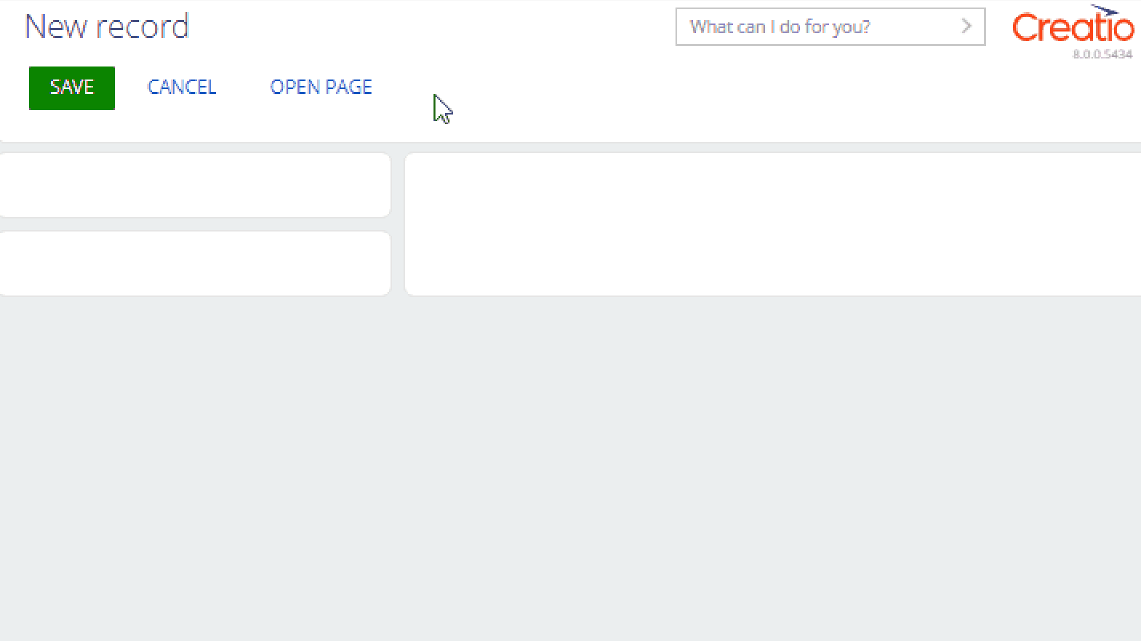
4. Click [*Save*] on the Client Module Designer's toolbar.

Outcome of the example

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

1. Open the `Handler Chain Service` app page and click [*Run app*].
2. Click [*New*] on the `Handler Chain Service` app toolbar.
3. Click [*Open Page*] on the record page of the custom `Handler Chain Service` section.

As a result, Creatio will open the `StudioHomePage` Freedom UI page.



Custom UI component based on a classic

Creatio page element

Beginner

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

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

- Custom component based on a classic Creatio page element. Supported in Creatio 8.0.2 Atlas and later.
- Remote module. Supported in Creatio 8.0.3 Atlas and later.

You can implement a custom component based on a classic Creatio page element in Creatio version 8.0.2 Atlas and later.

To **implement a custom component based on a classic Creatio page element**:

1. Create a custom component.
2. Add the custom component to the Freedom UI page.

1. Create a custom component

1. Create a module schema. To do this, follow the instruction in a separate article: Client module.
2. Implement a **custom component**.
   a. Add a dependency on the `@creatio-devkit/common` library to the AMD module.

Example that adds a dependency to the `UsrAppClientSchemaName` AMD module

```
/* AMD module declaration. */
define("UsrAppClientSchemaName", ["@creatio-devkit/common"], function (sdk) {
    ...
});
```

   b. Declare the component class.

Example that declares the `UsrAppClientSchemaName` class

```
/* AMD module declaration. */
define("UsrAppClientSchemaName", ["@creatio-devkit/common"], (sdk) {
    /* Declare the class. */
    class UsrAppClientSchemaName extends HTMLElement {
        constructor() {
            super();
            const shadowDom = this.attachShadow({mode: 'open'});
```

```
                shadowDom.innerHTML = '<h1>UsrAppClientSchemaName works!</h1>'
            }
        }
        ...
});
```

c. Register the component.

Example that registers the `UsrAppClientSchemaName` component

```
/* AMD module declaration. */
define("UsrAppClientSchemaName", ["@creatio-devkit/common"], (sdk) {
    ...
    /* Register the component. */
    customElements.define('usr-custom-view-element', UsrAppClientSchemaName);
    ...
});
```

d. Register the web component as a view element.

Example that registers the `usr-custom-view-element` component

```
/* AMD module declaration. */
define("UsrAppClientSchemaName", ["@creatio-devkit/common"], (sdk) {
    ...
    /* Register the web component as a view element. */
    sdk.registerViewElement({
        type: 'usr.CustomViewElement',
        selector: 'usr-custom-view-element'
    });
});
```

2. Add the custom component to the Freedom UI page

1. Add the custom component module to the AMD module declaration as a dependency.

Example that adds the `UsrAppClientSchemaName` dependency

```
/* AMD module declaration. */
define("UsrAppClientSchemaName1", ["UsrAppClientSchemaName"], () {
    ...
});
```

2. Add the configuration object of the module that contains the custom component to the `viewConfigDiff`

schema section.

View the configuration object of the module that contains the `UsrCustomViewElement` custom component below.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "insert",
        "name": "UsrAppClientSchemaName",
        "values": {
            "type": "usr.CustomViewElement",
            "layoutConfig": {
                "column": 1,
                "row": 1,
                "colSpan": 3,
                "rowSpan": 1
            }
        },
        "parentName": "Main",
        "propertyName": "items",
        "index": 0
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

Note. It is not possible to add a custom component based on a classic Creatio page element to the component library of the Freedom UI Designer. The Freedom UI Designer displays a placeholder instead of the custom component on the canvas. This is because classic Creatio page elements are implemented using the ExtJs framework. To display a custom component in the element library of the Freedom UI Designer, implement a remote module. To do this, follow the instruction in a separate article: Implement a remote module.

View a detailed example that implements the custom component in a separate article: Implement a custom component based on a classic Creatio page.

Implement a custom component based on a classic Creatio page

Medium

The example is relevant to version 8.0.2 and later.

Example. Display the history of the selected account on the [*Account timeline*] tab of the record page of the [*Requests*] custom section. The tab is a web component. Implement the web component based on the 7.X [*Timeline*] tab displayed on the contact page.

1. Create an app

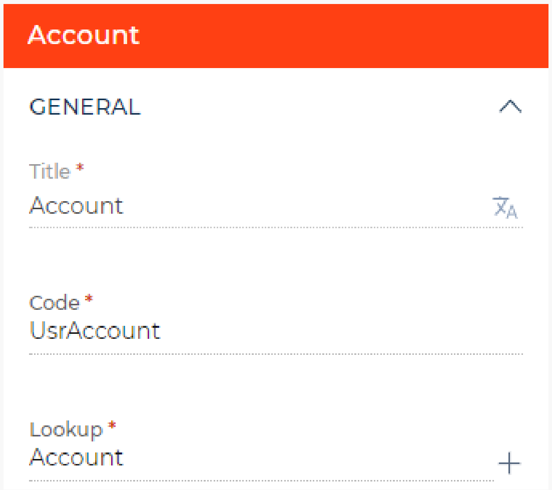
1. Create a custom `Requests` app based on the [*Records & business processes*] template. To do this, follow the procedure in the user documentation: Create a custom app.
2. Open the [*Requests form page*] page in the workspace of the `Requests` app page.

   The [*Requests form page*] page includes the [*Name*] field by default.
3. Add an **account field**:

   a. Add a new field of the [*Dropdown*] type to the Freedom UI Designer's workspace.

   b. Click the button on the Freedom UI Designer's action panel and fill out the **field properties** in the setup area:

   - Set [*Title*] to "`Account`."
   - Set [*Code*] to "`UsrAccount`."
   - Select "`Account`" in the [*Lookup*] property.
   - Clear the [*Enable adding new values*] checkbox.



Account

GENERAL

Title *
Account

Code *
UsrAccount

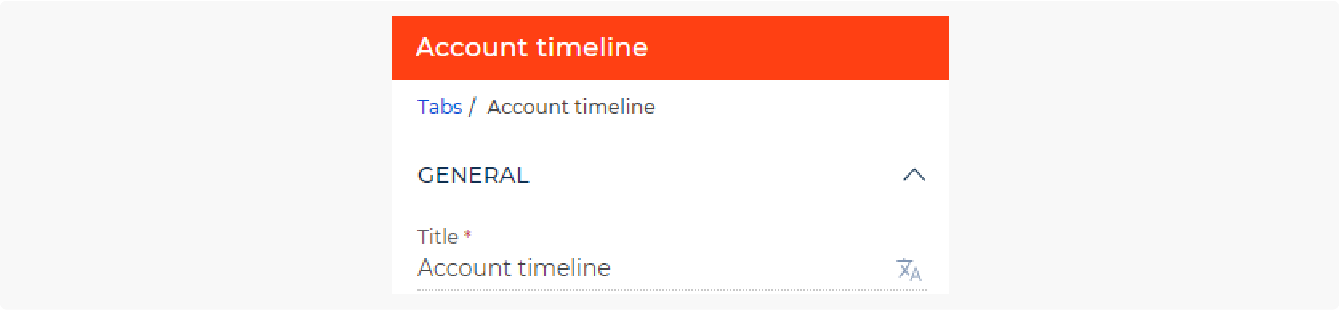
Lookup *
Account

4. Add a **tab that contains the history of the selected account**.

   a. Add a new [*Tabs*] layout element to the Freedom UI Designer's workspace.

   b. Delete the [*Tab 2*] tab the [*Requests form page*] page includes by default.

   c. Click the button on the Freedom UI Designer's action panel and specify "`Account timeline`" in the [*Title*] **tab property** in the setup area.



Account timeline

Tabs / Account timeline

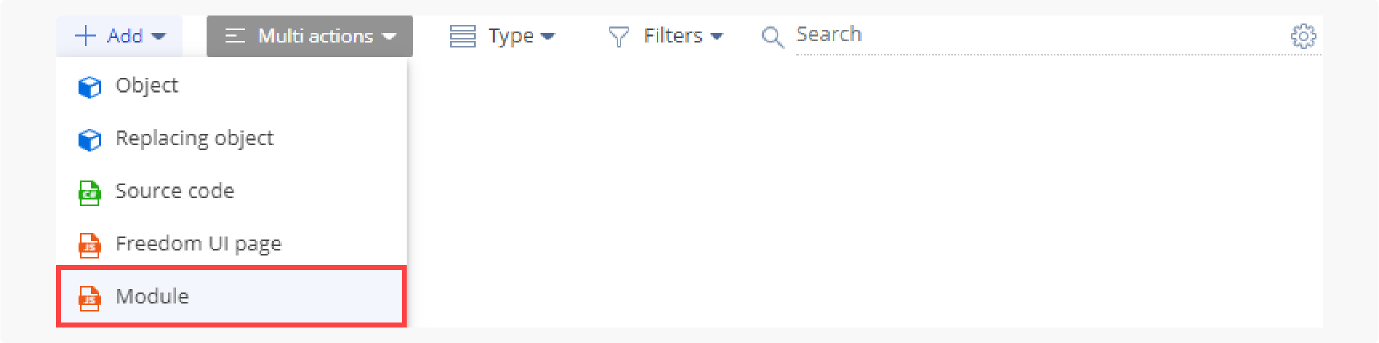
GENERAL

Title *
Account timeline

5. Click [*Save*] on the Freedom UI Designer's action panel.

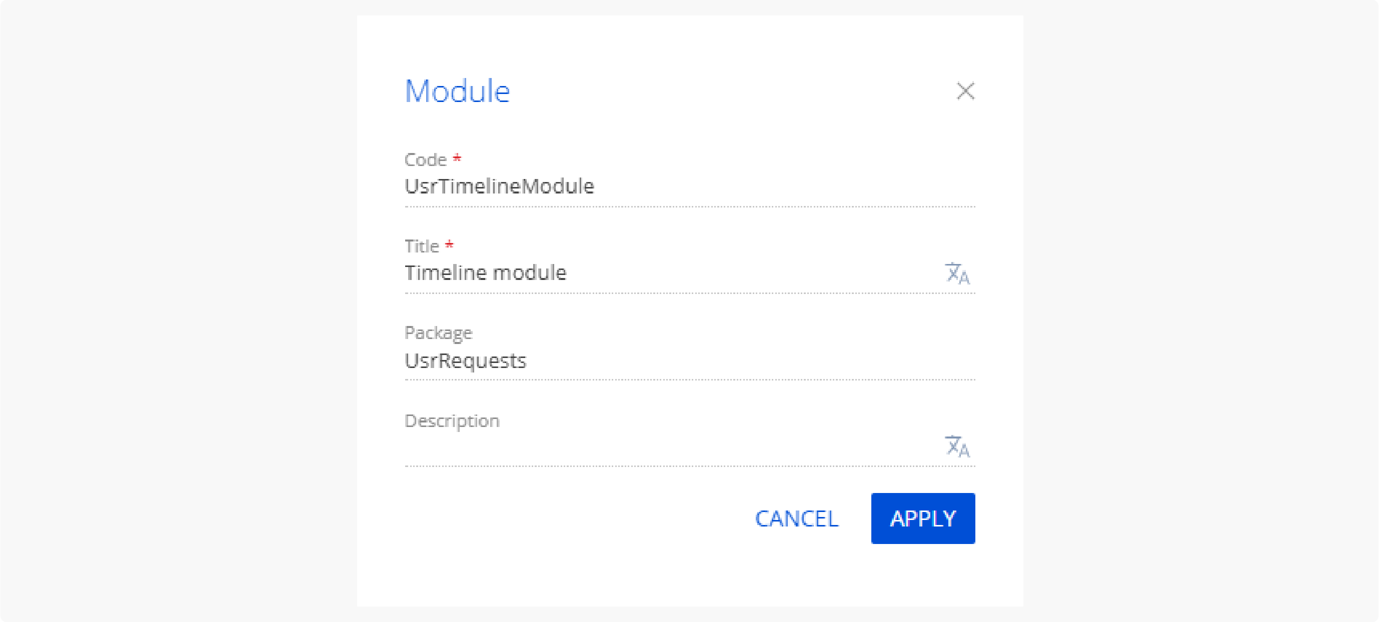
2. Create a custom web component

1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Module*] on the section list toolbar.



3. Fill out the **schema properties** in the Module Designer.

- Set [*Code*] to "`UsrTimelineModule`."
- Set [*Title*] to "`Timeline module`."



Module

Code *
UsrTimelineModule

Title *
Timeline module

Package
UsrRequests

Description

4. Implement a **custom web component**.

    a. Add the `@creatio-devkit/common`, `Base7xViewElement`, and `ckeditor-base` modules as dependencies to the AMD module declaration.

    Dependencies of the `UsrTimelineModule` AMD module

    ```
    /* AMD module declaration. */
    define("UsrTimelineModule", ["@creatio-devkit/common", "Base7xViewElement", "ckeditor-base"
        ...
    });
    ```

    b. Declare a custom `UsrTimelineModule` web component class.

    `UsrTimelineModule` class declaration

    ```
    /* AMD module declaration. */
    define("UsrTimelineModule", ["@creatio-devkit/common", "Base7xViewElement", "ckeditor-base"
        /* Class declaration. */
        class UsrTimelineModule extends Base7xViewElement {
            set primaryColumnValue(value) {
                this._primaryColumnValue = value;
                this._init();
            }

            get primaryColumnValue() {
                return this._primaryColumnValue;
            }

            set entitySchemaName(value) {
                this._entitySchemaName = value;
                this._init();
            }

            get entitySchemaName() {
                return this._entitySchemaName;
            }

            set cardSchemaName(value) {
                this._cardSchemaName = value;
                this._init();
            }

            get cardSchemaName() {
                return this._entitySchemaName;
            }
    ```

```
        constructor() {
            super("Timeline");
        }

        _init() {
            if (this._primaryColumnValue && this._cardSchemaName && this._entitySchemaName)
                this.initContext(() => {
                    this._moduleId = this.sandbox.id + "_UsrTimelineModule";
                    this.sandbox.subscribe("GetColumnsValues", (attributeNames) => this._ge
                    this.sandbox.subscribe("GetEntityInfo", () => this._getEntityInfo(), nu
                    this._loadTimelineSchemaModule();
                });
            }
        }

        _loadTimelineSchemaModule() {
            this._moduleId = this.sandbox.loadModule("BaseSchemaModuleV2", {
                id: this._moduleId,
                renderTo: this._renderTo,
                instanceConfig: {
                    schemaName: "TimelineSchema",
                    isSchemaConfigInitialized: true,
                    useHistoryState: false,
                    showMask: true,
                    parameters: {
                        viewModelConfig: {
                            "CardSchemaName": this._cardSchemaName,
                            "ReferenceSchemaName": this._entitySchemaName,
                            "InitialConfig": {
                                "entities": []
                            }
                        },
                    },
                }
            });
        }

        _getColumnValues(attributeNames) {
            const values = {};
            attributeNames?.forEach((attributeName) => {
                switch (attributeName) {
                    case "Id":
                        values[attributeName] = this._primaryColumnValue?.value;
                        break;
                    case "Name":
                        values[attributeName] = this._primaryColumnValue?.displayValue;
                        break;
                    default: break;
                }
            });
            return values;
        }

        _getEntityInfo() {
            return {
                entitySchemaName: this._entitySchemaName,
                primaryColumnValue: this._primaryColumnValue?.value,
                primaryDisplayColumnValue: this._primaryColumnValue?.displayValue
            };
        }

        getMessages() {
            const messages = super.getMessages();
            return Object.assign(messages, {
                "GetColumnsValues": {
                    mode: Terrasoft.MessageMode.PTP,
                    direction: Terrasoft.MessageDirectionType.SUBSCRIBE
                },
                "GetEntityInfo": {
                    mode: Terrasoft.MessageMode.PTP,
                    direction: Terrasoft.MessageDirectionType.SUBSCRIBE
                }
            });
        }

        disconnectedCallback() {
            this.sandbox.unloadModule(this._moduleId, this._renderTo);
        }
    }
    ...
});
```

c. Register the `UsrTimelineModule` web component on the page.

Register the `UsrTimelineModule` web component

```
/* AMD module declaration. */
define("UsrTimelineModule", ["@creatio-devkit/common", "Base7xViewElement", "ckeditor-base"
    ...
    /* Web component registration. */
    customElements.define('usr-timeline', UsrTimelineModule);
    ...
});
```

d. Register the `usr-timeline` web component as a visual element.

`usr-timeline` **web component registration**

```
/* AMD module declaration. */
define("UsrTimelineModule", ["@creatio-devkit/common", "Base7xViewElement", "ckeditor-base
    ...
    /* Register a web component as a visual element. */
    sdk.registerViewElement({
        type: 'usr.Timeline',
        selector: 'usr-timeline',
        inputs: {
            primaryColumnValue: {},
            cardSchemaName: {},
            entitySchemaName: {}
        }
    });
});
```

Complete source code of the page schema

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

3. Add the custom web component to the Freedom UI page

1. Open the `UsrRequests_FormPage` schema of the Freedom UI [*Requests form page*] page in the [*Configuration*] section.
2. Click the </> button on the Freedom UI Designer’s action panel. This opens the source code of the Freedom UI page.
3. Add the **custom web component**.

    a. Add the `UsrTimelineModule` module of the custom web component as a dependency to the AMD module declaration.

Dependencies of the `UsrRequests_FormPage` **AMD module**

```
/* AMD module declaration. */
define("UsrRequests_FormPage", /**SCHEMA_DEPS*/["UsrTimelineModule"]/**SCHEMA_DEPS*/, funct
    ...
});
```

    b. Add the configuration object of the `UsrTimelineModule` module with the custom web component to the `viewConfigDiff` schema section.

`viewConfigDiff` **schema section**

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "Timeline_qwerty",
        "values": {
            "type": "usr.Timeline",
            "layoutConfig": {
                "column": 1,
                "row": 1,
                "colSpan": 12,
                "rowSpan": 8
            },
            "primaryColumnValue": "$UsrAccount",
            "cardSchemaName": "AccountPageV2",
            "entitySchemaName": "Account"
        },
        "parentName": "GridContainer_qaocexw",
        "propertyName": "items",
        "index": 0
    },
    ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

Complete source code of the page schema

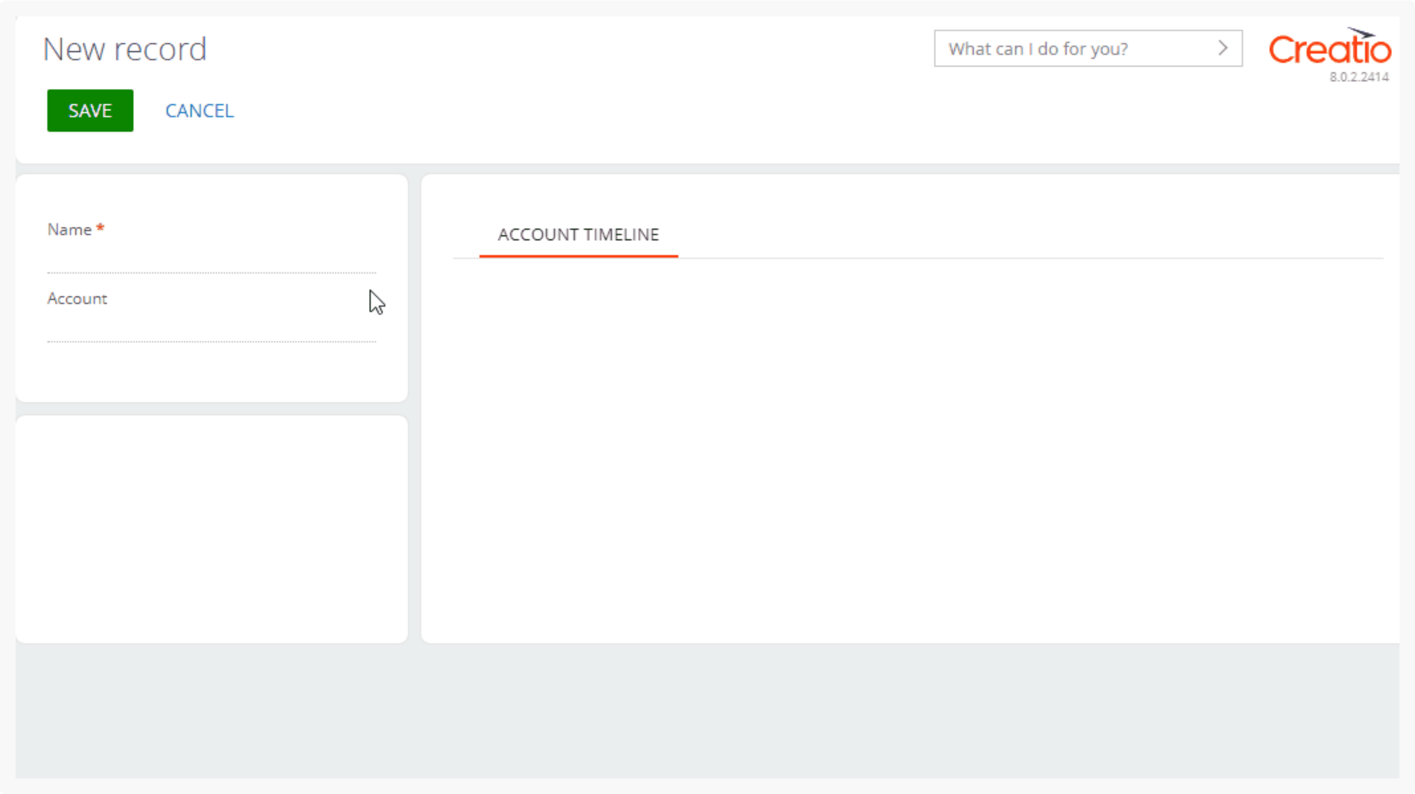
4. Click [*Save*] on the Client Module Designer’s toolbar.

Outcome of the example

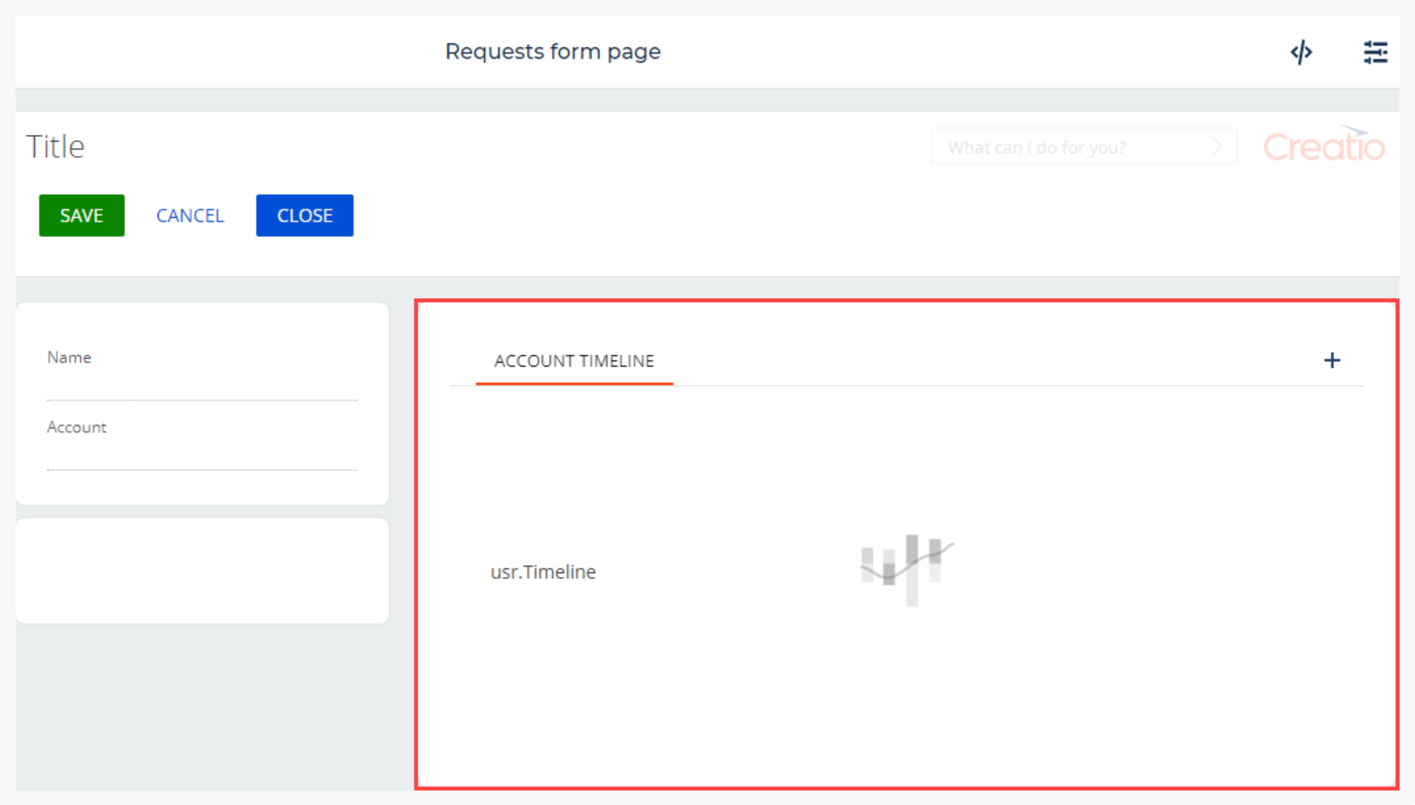
To **view the outcome of the example**:

1. Open the `Requests` app page and click [*Run app*]
2. Click [*New*] on the `Requests` app toolbar.
3. Enter "`Request’s name`" in the [*Name*] field.
4. Select an account in the [*Account*] field, for example, "`Accom`."

As a result, Creatio will display the [*Account timeline*] tab of the selected "`Accom`" account on the request page.



Freedom UI Designer will display a stub in place of the custom web component.



Set up a custom action menu for list and

list records

Easy

You can set up a custom action menu for the list and list records in Creatio version 8.0.4 Atlas and later.

Creatio lets you use action menu that contains default actions for the list and list records or set up a custom action menu.

By default, the action menu for the list contains the following **actions**:

- [*Save list settings for all users*]
- [*Reset to default list settings*]

By default, the action menu for list records contains the following **actions**:

- [*Open*]
- [*Copy*]
- [*Delete*]

Set up a custom action menu

To **set up a custom action menu for the list and list records**:

1. Open the [*Freedom UI page*] schema that contains the list.
2. Go to the `viewConfigDiff` schema section and find the configuration object whose the `type` property value is "`crt.DataGrid`". The `DataGrid` component implements the [*List*] type component in the front-end.
3. Implement a custom action menu.

    - Add the `headerToolbarItems` array of properties to implement a custom action menu **for the list**.
    - Add the `rowToolbarItems` array of properties to implement a custom action menu **for list records**.

    Attention. Add a configuration object to the property array for each action menu item.

    To add an item to the action menu, use the following **properties**:

    - `type`. The registered name of the component to display in the action menu item. Use the "`crt.MenuItem`" value.
    - `caption`. The text to display in the action menu item. Can be a constant or "`#ResourceString(SomeLocalizableString)#`" macro. `SomeLocalizableString` is the localizable string name in the schema of the Freedom UI page.
    - `icon`. The name of the icon to display in the action menu item, for example, "`edit-row-action`". We recommend using the `icon` property values of the icon button. Learn more in a separate article: Button component.

- `clicked`. The action to execute on item click. Learn more about the `clicked` property in a separate article: Open a page from a custom handler. Set the `useRelativeContext` property to `true` for the `rowToolbarItems` array. Do not use the `useRelativeContext` property for the `headerToolbarItems` array. The `useRelativeContext` property specifies the `ViewModel` of a page or list record to use in the query processing.

View the example of the `rowToolbarItems` array of properties below. This example implements a custom action menu for list records.

Example of the `rowToolbarItems` array of properties

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "merge",
        "name": "DataTable",
        "values": {
            ...,
            "rowToolbarItems": [{
                "type": 'crt.MenuItem',
                "caption": 'DataGrid.RowToolbar.Open',
                "icon": 'edit-row-action',
                "clicked": {
                    "request": 'crt.UpdateRecordRequest',
                    "params": {
                        "itemsAttributeName": "Items",
                        "recordId": "$Items.PDS_Id",
                    },
                    "useRelativeContext": true
                }
            }]
        }
        ...
    },
    ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/
```

As a result, the custom action menu for list records will contain only the [*Open*] item.

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

Close the WebSocket when destroying the View of the model

Medium

To **close the WebSocket when destroying** the `View` of the **model**, add a custom implementation of the `crt.HandleViewModelDestroyRequest` system query handler to the `handlers` schema section. The handler is executed when the `View` of the model is destroyed (for example, when you open another page). Designed to destroy resources. We do not recommend writing asynchronous code in the handler (server calls, timeouts, etc.) except for reading the value of attributes.

View an example of the `crt.HandleViewModelDestroyRequest` query handler that closes the custom `SomeWebSocket` WebSocket below.

For Creatio version **8.0.6 and later**

For Creatio version **8.0-8.0.5**