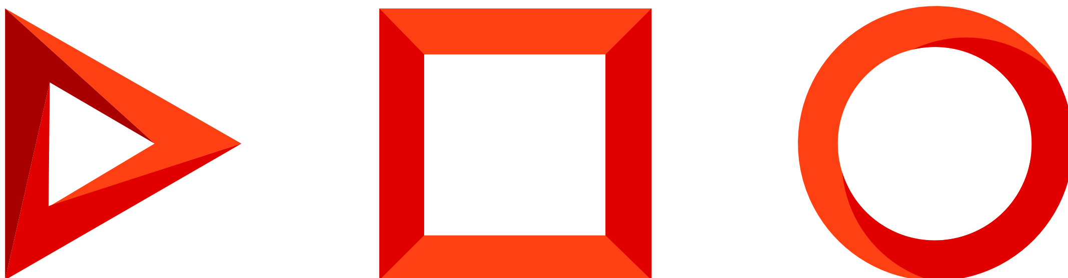
Creatio Academy

Front-end development

Client schema

Version 8.0



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

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

Table of Contents

Client schema 4

Develop a client schema 4

Client schema properties 5

Overload a mixin method 21

1. Create a mixin 21

2. Connect the mixin 23

3. Overload the mixin method 24

Method declaration example 25

Array of modifications usage example 26

Example of using the alias mechanism for repeated schema replacement 27

attributes property 30

Primary properties 30

Additional properties 33

messages property 35

Properties 35

rules and businessRules properties 36

Primary properties 36

Additional properties 39

diff property 42

Properties 42

© 2023 Creatio. All rights reserved.

Client schema

Easy

A **client view model schema** is a visual module schema that implements the front-end part of Creatio. A client view model schema is a configuration object for generating views and view models by `ViewGenerator` and `ViewModelGenerator`. Learn more about module types and their specificities in a separate article: Client module types.

Develop a client schema

Ways to develop client view model schemas:

- The [*Configuration*] section. Learn more about development in the [*Configuration*] section in a separate article: View model schema.
- Section Wizard. Learn more about section development in the Section Wizard in user documentation: Create a new section
- Detail Wizard. Learn more about detail development in the Detail Wizard in user documentation: Create a detail.

Structure elements of the client schema:

- Auto-generated code. Contains the description of the schema, its dependencies, localized resources, and messages.
- Rendering styles. Only available in some types of client schemas.
- The schema source code. A syntactically correct JavaScript code that defines the module.

Use **marker comments** in the schema source code for the `diff`, `modules`, `details`, and `businessRules` properties.

The **purpose** of marker comments is to uniquely identify the client schema properties. When you open the Wizard, Creatio validates the presence of marker comments as shown in the table below.

© 2023 Creatio. All rights reserved.

Validation rules for marker comments in client schemas

<table>
<tr><th>Schema type</th><th>Required marker comments</th></tr>
<tr><td><code>EditViewModelSchema</code> model view schema of a record page</td><td><code>details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/,
businessRules: /**SCHEMA_BUSINESS_RULES*/{}/**SCHEMA_BUSI</code></td></tr>
<tr><td><code>ModuleViewModelSchema</code> model view schema of a section</td><td rowspan="4"><code>modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/</code></td></tr>
<tr><td><code>EditControlsDetailViewModelSchema</code> model view schema of a detail with fields</td></tr>
<tr><td><code>DetailViewModelSchema</code> model view schema of a detail</td></tr>
<tr><td><code>GridDetailViewModelSchema</code> model view schema of a detail that has a list</td></tr>
</table>

Client schema properties

The source code of client schemas has a generic structure available below.

Source code of a client schema

```
define("ExampleSchema", [], function() {
    return {
        entitySchemaName: "ExampleEntity",
        mixins: {},
        attributes: {},
        messages: {},
        methods: {},
        rules: {},
        businessRules: /**SCHEMA_BUSINESS_RULES*/{}/**SCHEMA_BUSINESS_RULES*/,
        modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

© 2023 Creatio. All rights reserved.

After the module is loaded, Creatio calls the anonymous factory function, which returns the schema configuration object. **Properties** of the configuration object schema:

- `entitySchemaName`. The name of the entity schema used by the current client schema.
- `mixins`. A configuration object that contains a mixin declaration.
- `attributes`. A configuration object that contains schema attributes.
- `messages`. A configuration object that contains schema messages.
- `methods`. A configuration object that contains schema methods.
- `rules`. A configuration object that contains schema business rules.
- `businessRules`. A configuration object that contains schema business rules created or modified by the Section Wizard or Detail Wizard. The `/**SCHEMA_BUSINESS_RULES*/` marker comments are required since they are necessary for the operation of the Wizards.
- `modules`. A configuration object that contains schema modules. The `/ ** SCHEMA_MODULES * /` marker comments are required since they are necessary for the operation of the Wizards.

  > **Note.** The `details` property loads a detail to a page. Since a detail is also a module, we recommend using the `modules` property instead.

- `diff`. A configuration object array that contains the schema view description. The `/**SCHEMA_DIFF*/` marker comments are required since they are necessary for the operation of the Wizards.
- `properties`. A configuration object that contains view model properties.
- `$-properties`. Automatically generated properties for the attributes of the view model schema.

Schema name (`entitySchemaName`)

To implement the entity schema name, use the required `entitySchemaName` property. Simply specify it in one of the inheritance hierarchy schemas.

Example that declares the `entitySchemaName` property

```
define("ClientSchemaName", [], function () {
    return {
        /* Object schema (model). */
        entitySchemaName: "EntityName",
        /* ... */
    };
});
```

Mixins (`mixins`)

A mixin is a class that extends the functions of other classes. JavaScript does not support multiple inheritances. However, mixins let you extend the schema functionality without duplicating the logic used in the schema

© 2023 Creatio. All rights reserved.

methods. You can use the same set of actions in different client schemas of Creatio. Create a mixin to avoid duplicating the code in each schema. Mixins are **different** from other modules added to the dependency list in the way of calling their methods from the module schema. You can call to their methods directly, much like those of a schema. Use the `mixins` property to implement mixins.

Mixin management **procedure**:

1. Create a mixin.
2. Assign a name to the mixin.
3. Connect the corresponding name array.
4. Implement the mixin functionality.
5. Use the mixin in the client schema.

Create a mixin

Create a mixin similarly to an [object schema](#).

Assign a name to the mixin

When naming mixins, use an `-able` suffix in the schema name. For example, name a mixin that enables serializing in the components `Serializable`. If a mixin name cannot end "-able," end the schema name in `Mixin`.

> **Attention.** Do not use words like `Utilities`, `Extension`, `Tools`, or similar in the names. They make the purpose of the mixin impossible to discern based on the mixin name.

Connect the namespace

Enable a corresponding name array in the mixin (`Terrasoft.configuration.mixins` for the configuration, `Terrasoft.core.mixins` for the core).

Implement the mixin functionality

Mixins cannot depend on the internal implementation of the schema to which to apply them. Mixins must be independent mechanisms that receive a set of parameters, process them, and, if needed, return a result. Design mixins as modules that must be connected to the schema dependency list when the `define()` function declares the schema.

View the mixin structure below.

Mixin structure

```
define("MixinName", [], function() {
    Ext.define("Terrasoft.configuration.mixins.MixinName", {
        alternateClassName: "Terrasoft.MixinName",
        /* Mixin functionality. */
    });
    return Ext.create(Terrasoft.MixinName);
```

© 2023 Creatio. All rights reserved.

```
})
```

Use the mixin

The mixin implements the functionality needed in the client schema. To receive the set of mixin actions, specify the mixin in the `mixins` block of the client schema.

Use a mixin in the client schema

```
/* MixinName is a module where the mixin class is implemented. */
define("ClientSchemaName", ["MixinName"], function () {
    return {
        /* SchemaName is the name of the entity. */
        entitySchemaName: "SchemaName",
        mixins: {
            /* Connect the mixin. */
            MixinName: "Terrasoft.NameSpace.Mixin"
        },
        attributes: {},
        messages: {},
        methods: {},
        rules: {},
        modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

Once you connect the mixin, you can use its methods, attributes, and fields in the client schema as if they were part of the client schema. That way, method calls are more concise than when using a separate schema. For example, `getDefaultImageResource` is a mixin function. To call the `getDefaultImageResource` mixin function in the custom schema to which the mixin is connected, use `this.getDefaultImageResource();`.

Note. To overload a mixin function, create a function with the same name in the client schema. As a result, Creatio will use the function of the schema, and not that of the mixin, when calling.

Attributes (`attributes`)

Use the `attributes` property to implement attributes.

Messages (`messages`)

The **purpose** of messages is to organize data exchange between modules. Use the `messages` property to implement messages. Use the `Terrasoft.MessageMode` enumeration to set the message **mode**.

© 2023 Creatio. All rights reserved.

Message mode types

Message mode	Description	Connection
Address	Address messages are only received by the last subscriber.	To switch to address mode, set the `mode` property to `this.Terrasoft.MessageMode.PTP`.
Broadcasting	Broadcasting messages are received by all subscribers.	To switch to broadcasting `mode`, set the mode property to `this.Terrasoft.MessageMode.BROADCAST`.

Aside from modes, you can also specify the message **direction**.

Message direction types

Message direction	Description	Connection
Publishing	The message can only be published, i. e., it is an **outbound** message.	To set the message direction to publishing, set the `direction` property to `this.Terrasoft.MessageDirectionType.PUBLISH`.
Subscription	The message can only be subscribed to, i. e., it is an **inbound** message.	To set the message direction to subscription, set the `direction` property to `this.Terrasoft.MessageDirectionType.SUBSCRIBE`.
Bidirectional	The bidirectional mode enables publishing of and subscription to the same message in different instances of a single class or within a single schema inheritance hierarchy. The same message cannot be announced with different directions in a single schema inheritance hierarchy. Learn more about using bidirectional messages in cases where that is a requirement in a separate article: [Sandbox]().	Corresponds to `the Terrasoft.MessageDirectionType.BIDIRECTIONAL` enumeration value.

Message publication

Declare a message with the "`publishing`" direction in the schema where you want to publish the message.

© 2023 Creatio. All rights reserved.

Example that declares a message with the "`publishing`" direction

```
messages: {
    /* Message name. */
    "GetColumnsValues": {
        /* Set the message mode to address. */
        mode: this.Terrasoft.MessageMode.PTP,
        /* Set the message direction to "publishing." */
        direction: this.Terrasoft.MessageDirectionType.PUBLISH
    }
}
```

Publishing is done by calling the `publish` method from the `sandbox` class instance.

Message publishing example

```
// GetColumnsValues method that gets the message publishing result.
getColumnsValues: function(argument) {
/* Message publishing.
GetColumnsValues is the message name.
argument is the argument passed to the handler function of the subscriber. An argument is an obj
key is an array of message filtering tags. */
return this.sandbox.publish("GetColumnsValues", argument, ["key"]);
}
```

Attention. Message publishing can return the handler function results only in the **address** mode.

Message subscription

Declare a message with the "`subscription`" direction in the subscription schema.

Example that declares a message with the "`subscription`" direction

```
messages: {
    /* Message name. */
    "GetColumnsValues": {
        /* Set the message mode to address. */
        mode: this.Terrasoft.MessageMode.PTP,
        /* Set the message direction to "subscription." */
        direction: this.Terrasoft.MessageDirectionType.SUBSCRIBE
    }
}
```

© 2023 Creatio. All rights reserved.

The subscription is made by calling the `subscribe` method in the `sandbox` class instance.

Message subscription example

```
/* GetColumnsValues is the message name.
messageHandler is the message handler function.
context is the execution scope of the handler function.
key is an array of message filtering tags. */
this.sandbox.subscribe("GetColumnsValues", messageHandler, context, ["key"]);
```

In the **address** mode, the `messageHandler` method returns the object, which is processed as the result of message publishing. In **broadcasting** mode, the `messageHandler` method does not return a value.

messageHandler method (address mode)

```
methods: {
    messageHandler: function(args) {
        /* Return an object to process as a message publishing result. */
        return { };
    }
}
```

messageHandler method (broadcast mode)

```
methods: {
    messageHandler: function(args) {
    }
}
```

Methods(`methods`)

To implement a method, use the `methods` property.The property contains a collection of methods that form the schema business logic and affect the view model. By default, the scope of methods is the scope of view model.

The **purpose** of the `methods` property.

1. Create new methods.
2. Extend the basic methods of parent schemas.

Business rules (`rules` and `businessRules`)

Business rules are Creatio mechanisms that let you customize the behavior of fields on a page or detail. To

© 2023 Creatio. All rights reserved.

implement business rules, use the `rules` and `businessRules` properties. Use the `businessRules` property for business rules created or modified in the Section Wizard or the Detail Wizard.

The **purposes** of business rules:

- Hide or show fields.
- Lock or unlock fields for editing.
- Make fields required or optional.
- Filter lookup fields based on values in other fields.

Creatio implements the business rule functionality in the `BusinessRuleModule` client module. To use the business rule functionality, add `BusinessRuleModule` to the list of schema dependencies.

Example that adds `BusinessRuleModule` to the dependency list

```
define("CustomPageModule", ["BusinessRuleModule"],
    function(BusinessRuleModule) {
        return {
            /* Implement the client module. */
        };
    });
```

The `RuleType` enumeration of the `BusinessRuleModule` module defines business rule types.

Specifics of business rules

Specifics of business rule declaration:

- Describe business rules in the `rules` schema property.
- Apply business rules to view model columns and not to controls.
- Name each business rule.
- Set business rule parameters in the configuration object.

Business rules defined in the `businessRules` property have the following **features**:

- They are generated by the Section or Detail Wizards.
- When you create a new business rule, the corresponding Wizard generates the name and adds the rule to the client schema of the record page view model.
- Creatio does not use the `BusinessRuleModule` enumerations when describing generated business rules.
- The `/**SCHEMA_BUSINESS_RULES*/` marker comments are required since they are necessary for the operation of the Wizards.
- They have a higher priority during runtime.
- When a business rule is disabled, Creatio sets the `enabled` property of the configuration object to `false`.
- When a business rule is removed, the configuration object remains in the client schema of the record page

© 2023 Creatio. All rights reserved.

view model, but Creatio sets the `removed` property to `true`.

Attention. We do not recommend editing the `businessRules` property of the client schema.

Edit an existing business rule

After a Wizard edits a manually created business rule, the business rule's configuration object in the `rules` property of the record page view model remains unchanged. At the same time, a new version of the business rule configuration object with the same name is created in the `businessRules` property.

When Creatio processes a business rule during runtime, the business rule defined in the `businessRules` property takes precedence. Subsequent changes to this business rule in the `rules` property will not affect Creatio in any way.

Note. Changes made to the configuration object of the `businessRules` property take precedence when you delete or disable a business rule.

Modules (`modules`)

To implement modules, use the `modules` property. Its configuration object declares and configures modules and details loaded on the page. The `/ ** SCHEMA_MODULES * /` marker comments are required since they are necessary for the operation of the Wizards.

Note. The `details` property loads a detail to a page. Since a detail is also a module, we recommend using the `modules` property instead.

Example that uses the `modules` property

```
modules: /**SCHEMA_MODULES*/{
    /* Load the module.
    Module title. Must be the same as the name property in the diff array. */
    "TestModule": {
        /* Optional. The ID of the module to load. If not specified, Creatio will generate it au
        "moduleId": "myModuleId",.
        /* If no parameter is specified, Creatio will use BaseSchemaModuleV2 for loading. */
        "moduleName": "MyTestModule",
        /* Configuration object. When the module is loaded, the object is passed as instanceConf
        "config": {
            "isSchemaConfigInitialized": true,
            "schemaName": "MyTestSchema",
            "useHistoryState": false,
            /* Additional module parameters. */
            "parameters": {
```

© 2023 Creatio. All rights reserved.

```
                /* Parameters passed to the schema during the initialization. */
                "viewModelConfig": {
                    masterColumnName: "PrimaryContact"
                }
            }
        }
    },

    /* Load the detail.
    Detail name. */
    "Project": {
            /* Detail schema name. */
            "schemaName": "ProjectDetailV2",
            "filter": {
            /* The column of the section object schema. */
            "masterColumn": "Id",
            /* The column of a detail object schema. */
            "detailColumn": "Opportunity"
        }
    }
}/**SCHEMA_MODULES*/
```

Array of modifications (`diff`)

To implement an array of modifications, use the `diff` property, which contains an array of configuration objects. The **purpose** of the array of modifications is to build a representation of the module in the Creatio interface. Each element in the array represents metadata from which Creatio generates various interface controls. The `/**SCHEMA_DIFF*/` marker comments are required since they are necessary for the operation of the Wizards.

The alias mechanism

When developing new versions, you sometimes need to move page elements to new zones. In situations where users have customized the record page, such changes can have unpredictable consequences. The `alias` **mechanism** interacts with the `diff` builder to provide partial backward compatibility when changing the UI in new product versions. The builder is the `json-applier` class that merges base schema and client extension schema parameters.

The `alias` property contains data about the previous name of the element. Creatio creates the `diff` array of modifications based on that data, considering not only elements with a new name but also with the name specified in `alias`. In essence, `alias` is a configuration object that links the new and old elements. When creating a `diff` array of modifications, the `alias` configuration object can disallow application of some properties and operations to the element where it is declared. You can add the `alias` object to any element in the `diff` array of modifications.

Relationship between a view and model

The **purpose** of the `bindTo` property is to indicate the relationship between a view model attribute and a view

© 2023 Creatio. All rights reserved.

object property.

Declare the property in the `values` property of the configuration objects in the `diff` array of modfications.

View an example that uses the `bindTo` property below.

Example that uses the `bindTo` property

```
diff: [
    {
        "operation": "insert",
        "parentName": "CombinedModeActionButtonsCardLeftContainer",
        "propertyName": "items",
        "name": "MainContactButton",
        /* Properties passed to the component’s constructor. */
        "values": {
            /* Set the type of the added element to button. */
            "itemType": Terrasoft.ViewItemType.BUTTON,
            /* Bind the button title to the localizable schema string. */
            "caption": {bindTo: "Resources.Strings.OpenPrimaryContactButtonCaption"},
            /* Bind the button click handler method. */
            "click": {bindTo: "onOpenPrimaryContactClick"},
            /* The display style of the button. */
            "style": Terrasoft.controls.ButtonEnums.style.GREEN,
            /* Bind the button availability property. */
            "enabled": {bindTo: "ButtonEnabled"}
        }
    }
]
```

Tabs are objects that contain the `tabs` value in their `propertyName` property.

Creatio implements an alternative way to use the `bindTo` property for **tab titles**.

Alternative way to use the `bindTo` property

```
...
{
    "operation": "insert",
    "name": "GeneralInfoTab",
    "parentName": "Tabs",
    /* Imply that the object is a tab. */
    "propertyName": "tabs",
    "index": 0,
    "values": {
        /* $ replaces usage of bindTo: {...}. */
        "caption": "$Resources.Strings.GeneralInfoTabCaption",
        "items": []
```

© 2023 Creatio. All rights reserved.

```
    }
},
...
```

`diff` property declaration rules

- **Best use of converters**.

  A **converter** is a function executed in the `viewModel` environment. A converter accepts the values of the `viewModel` property and returns a result of the corresponding type. To ensure that Wizards operate correctly, format the `diff` property value in JSON. Therefore, the value of the converter must be the name of the view model method rather than an inline function.

  ```
  Correct use of the converter

  methods: {
      someFunction: function(val) {
          /* ... */
      }
  },

  diff: /**SCHEMA_DIFF*/[
    {
        /* ... */
        "bindConfig": {
            "converter": "someFunction"
        }
        /* ... */
    }
  ]/**SCHEMA_DIFF*/
  ```

  ```
  Incorrect use of the converter

  diff: /**SCHEMA_DIFF*/[
      {
          /* ... */
          "bindConfig": {
              "converter": function(val) {
              /* ... */
              }
          }
      }
  ]/**SCHEMA_DIFF*/
  ```

© 2023 Creatio. All rights reserved.

Correct use of the generator

```
methods: {
    someFunction: function(val) {
        /* ... */
    }
},

diff: /**SCHEMA_DIFF*/[
  {
      /* ... */
      "values": {
          "generator": "someFunction"
      }
      /* ... */
  }
]/**SCHEMA_DIFF*/
```

Incorrect use of the generator

```
diff: /**SCHEMA_DIFF*/[
    {
        /* ... */
        "values": {
            "generator": function(val) {
                /* ... */
            }
        }
    }
]/**SCHEMA_DIFF*/
```

- **Parent element**.

  The **parent element (container)** is the DOM element where the module renders its view. To ensure that the Wizards operate as intended, place a single child element in the parent container.

Example of the correct placement of a view in the parent element

```
<div id="OpportunityPageV2Container" class="schema-wrap one-el" data-item-marker="Opportunity
    <div id="CardContentWrapper" class="card-content-container page-with-left-el" data-item-m
</div>
```

© 2023 Creatio. All rights reserved.

Example of incorrect placement of a view in the parent element

```
<div id="OpportunityPageV2Container" class="schema-wrap one-el" data-item-marker="Opportunity
    <div id="CardContentWrapper" class="card-content-container page-with-left-el" data-item-m
    <div id="DuplicateContainer" class="DuplicateContainer"></div>
</div>
```

When you add, change, move an element (`insert`, `merge`, `move` operations), specify the `parentName` property (the parent element's name) in the `diff` property.

Example of the correct definition of the view element in the diff property

```
{
  "operation": "insert",
  "name": "SomeName",
  "propertyName": "items",
  "parentName": "SomeContainer",
  "values": {}
}
```

Example of incorrect definition of the view element in the diff property

```
{
  "operation": "insert",
  "name": "SomeName",
  "propertyName": "items",
  "values": {}
}
```

If the `parentName` property is missing, the Wizard will be unable to configure the page. Creatio will display a corresponding error message.

The value of the `parentName` property must match the name of the parent element in the corresponding base page schema. For example, this is `CardContentContainer`for record pages.

If you specify the name of a non-existent container element as the parent element in the `parentName` property, a "`Schema cannot have more than one root object`" error will occur, since the added element will be placed in the root container.

- **Unique names.**

  Each element in the `diff` array must have a unique name.

Example of the correct addition of elements to a diff array

© 2023 Creatio. All rights reserved.

```
{
  "operation": "insert",
  "name": "SomeName",
  "values": { }
},
{
  "operation": "insert",
  "name": "SomeSecondName",
  "values": { }
}
```

Example of incorrect addition of elements to a diff array

```
{
  "operation": "insert",
  "name": "SomeName",
  "values": { }
},
{
  "operation": "insert",
  "name": "SomeName",
  "values": { }
}
```

- **Placement of view elements**.

  To ensure the view elements are customizable and changeable, place them on the **layout grid**. In Creatio, each row of the layout grid has 24 cells (columns). Use the `layout` property to place elements on the grid.

  Grid element **properties**:

  - `column`. The index of the left column.
  - `row`. The index of the top row.
  - `colSpan`. The number of spanned columns.
  - `rowSpan`. The number of spanned rows.

  Example that places elements

```
{
  "operation": "insert",
  "parentName": "ParentContainerName",
  "propertyName": "items",
  "name": "ItemName",
  "values": {
```

© 2023 Creatio. All rights reserved.

```
        /* Element placement. */
        "layout": {
            /* Start at column zero. */
            "column": 0,
            /* Place in the fifth row of the grid. */
            "row": 5,
            /* Span 12 columns wide. */
            "colSpan": 12,
            /* Occupy one row. */
            "rowSpan": 1
        },
        "contentType": Terrasoft.ContentType.ENUM
      }
    }
```

- **Number of operations**.

  If you change the client schema without a Wizard, we recommend adding no more than one operation per schema element to ensure that the Wizard operates as intended.

Properties (`properties`)

To implement properties, use the `properties` property, which contains a JavaScript object.

View an example that uses the `properties` property in the `SectionTabsSchema` schema of the `NUI` package below.

Example that uses the `properties` property

```
define("SectionTabsSchema", [],
    function() {
        return {
            ...
            /* Declare the properties property. */
            properties: {
                /* The parameters property. Array. */
                parameters: [],
                /* modulesContainer property. Entity. */
                modulesContainer: {}
            },
            methods: {
                ...
                /* Initialization method. Always executed first. */
                init: function(callback, scope) {
                    ...
                    /* Call the method uses the view model properties. */
                    this.initContainers();
                    ...
                },
```

© 2023 Creatio. All rights reserved.

```
                ...
                /* The method where to use the properties. */
                initContainers: function() {
                    /* Use the modulesContainer property. */
                    this.modulesContainer.items = [];
                    ...
                    /* Use the parameters property. */
                    this.Terrasoft.each(this.parameters, function(config) {
                        config = this.applyConfigs(config);
                        var moduleConfig = this.getModuleContainerConfig(config);
                        var initConfig = this.getInitConfig();
                        var container = viewGenerator.generatePartial(moduleConfig, initConfig)[
                        this.modulesContainer.items.push(container);
                    }, this);
                },
                ...
            },
        ...
        }
    });
```

Overload a mixin method

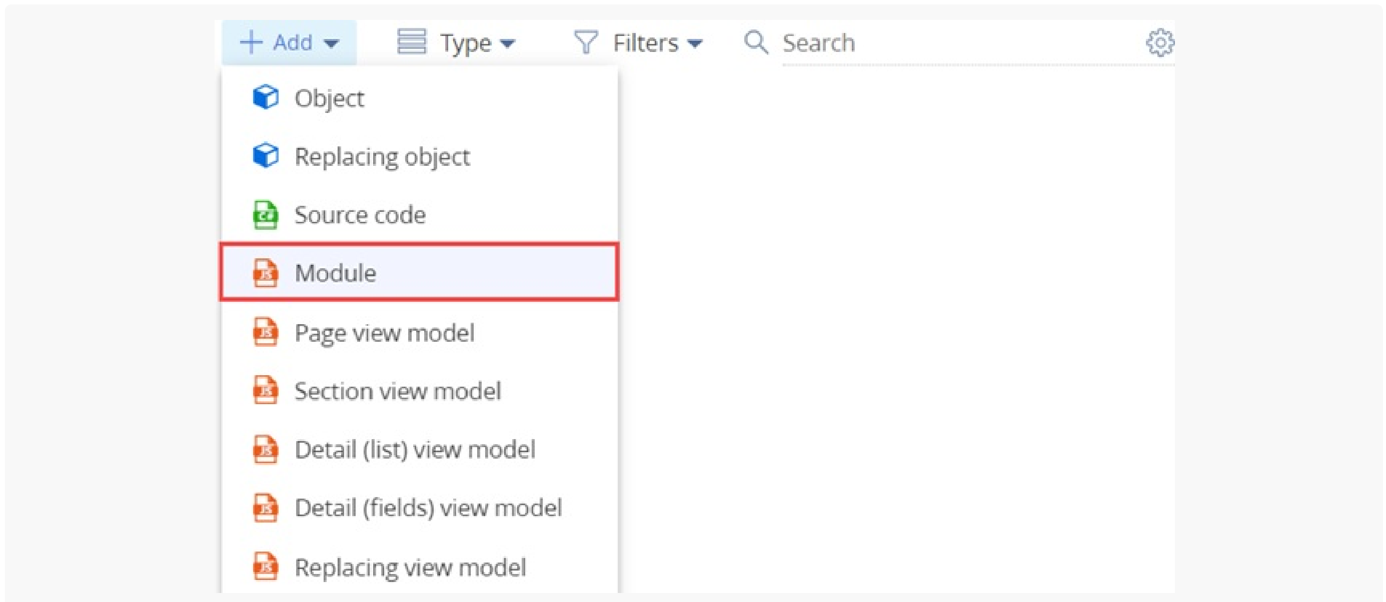
Beginner

Example. Connect the `ContentImageMixin` mixin to the custom schema, override the mixin method.

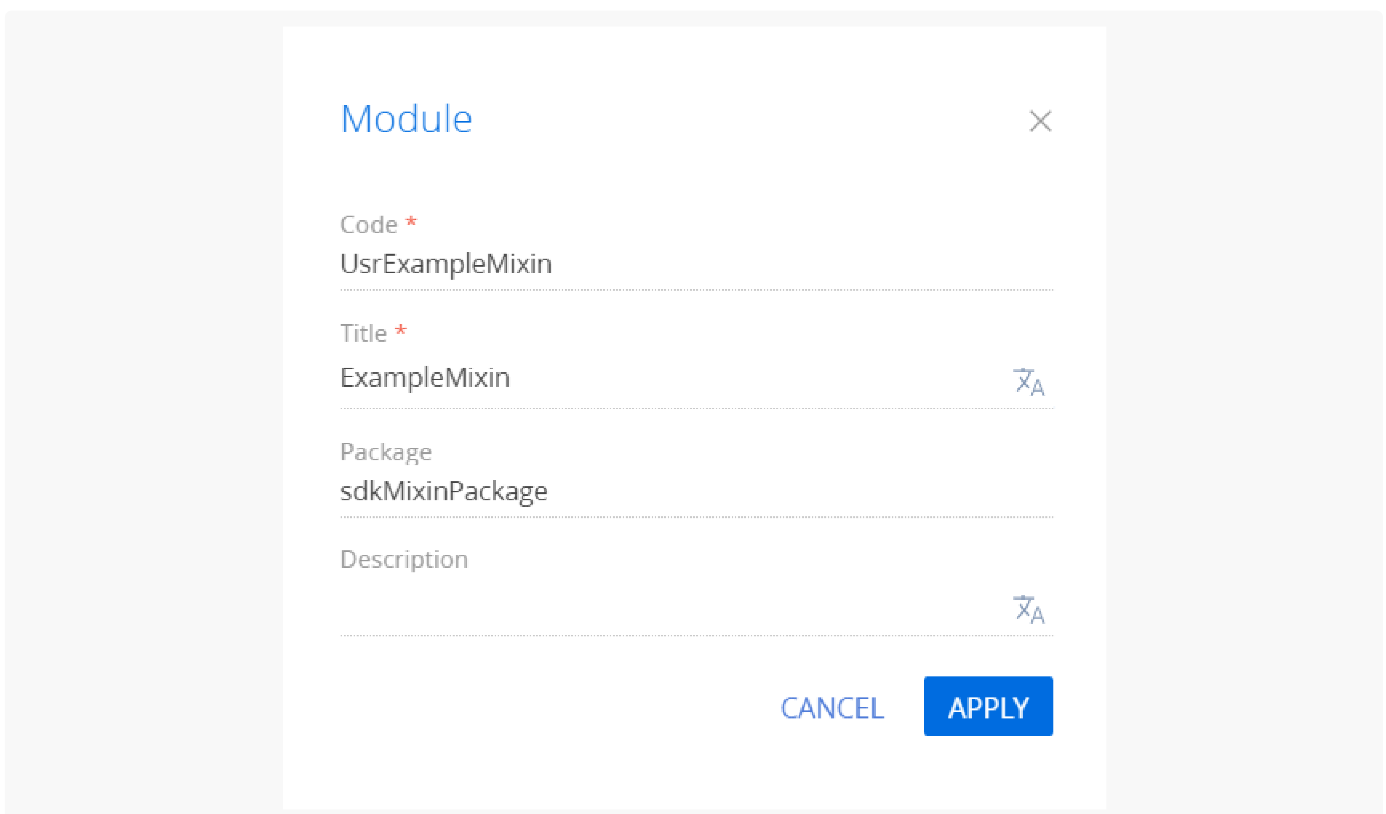
1. Create a mixin

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

© 2023 Creatio. All rights reserved.



3. Fill out the schema properties in the Schema Designer.
   - Set [*Code*] to "`UsrExampleMixin`."
   - Set [*Title*] to "`ExampleMixin`."



Click [*Apply*] to apply the properties.

4. Add the source code in the Schema Designer.

Module source code

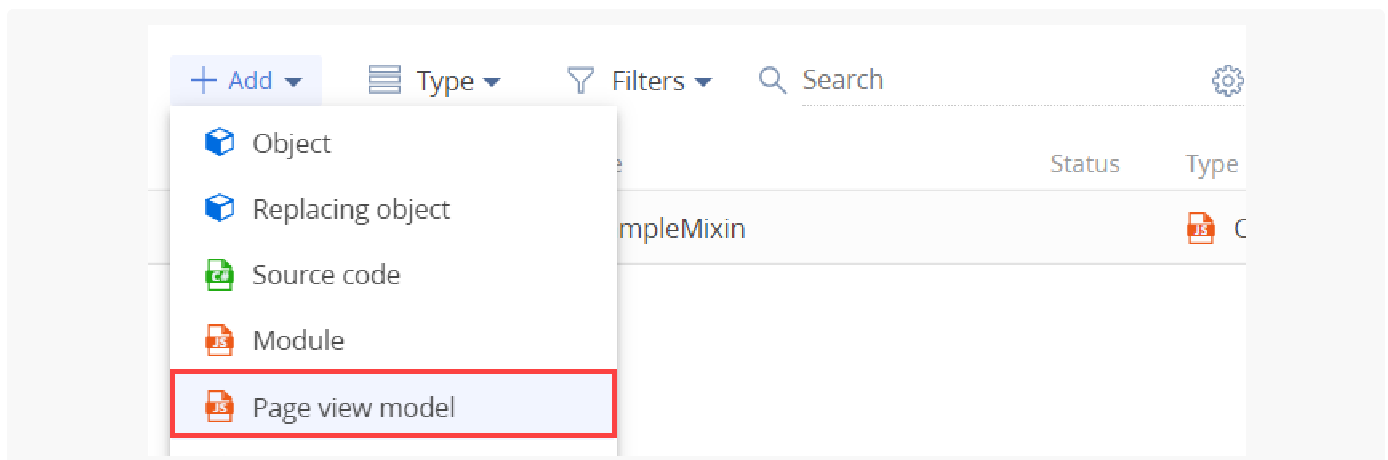
© 2023 Creatio. All rights reserved.

```
/* Define the module. */
define("ContentImageMixin", [ContentImageMixinV2Resources], function() {
    /* Define the ContentImageMixin class. */
    Ext.define("Terrasoft.configuration.mixins.ContentImageMixin", {
        /* Alias (shorthand for the class name). */
        alternateClassName: "Terrasoft.ContentImageMixin",
        /* Mixin functionality. */
        getImageUrl: function() {
            var primaryImageColumnValue = this.get(this.primaryImageColumnName);
            if (primaryImageColumnValue) {
                return this.getSchemaImageUrl(primaryImageColumnValue);
            } else {
                var defImageResource = this.getDefaultImageResource();
                return this.Terrasoft.ImageUrlBuilder.getUrl(defImageResource);
            }
        }
    });
    return Ext.create(Terrasoft.ContentImageMixin);
});
```

5. Click [*Save*] on the Designer toolbar.

2. Connect the mixin

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



3. Fill out the schema properties in the Schema Designer.

- Set [*Code*] to "`UsrExampleSchema`."
- Set [*Title*] to "`ExampleSchema`."
- Set [*Parent object*] to "`BaseProfileSchema`."

© 2023 Creatio. All rights reserved.

Module

Code *
UsrExampleSchema

Title *
ExampleSchema

Parent object *
BaseProfileSchema (BaseProfileSchema)

Package
sdkMixinPackage

Description

CANCEL APPLY

Click [*Apply*] to apply the properties.

To use the mixin, enable it in the `mixins` block of the `ExampleSchema` custom schema.

3. Overload the mixin method

Add the source code in the Schema Designer. In the `method` block, override the `getReadImageURL()` mixin method. Use the overridden function in the `diff` block.

Module source code

```
/* Declare the module. Include as a dependency the ContentImageMixin module, in which the mixin
define("UsrExampleSchema", ["ContentImageMixin"], function() {
    return {
        entitySchemaName: "ExampleEntity",
        mixins: {
            /* Connect the mixin to the schema. */
            ContentImageMixin: "Terrasoft.ContentImageMixin"
        },
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        diff: /**SCHEMA_DIFF*/[
            {
                "operation": "insert",
```

© 2023 Creatio. All rights reserved.

```
                "parentName": "AddRightsItemsHeaderImagesContainer",
                "propertyName": "items",
                "name": "AddRightsReadImage",
                "values": {
                    "classes": {
                        "wrapClass": "rights-header-image"]
                    },
                    "getSrcMethod": "getReadImageUrl",
                    "imageTitle": resources.localizableStrings.ReadImageTitle,
                    "generator": "ImageCustomGeneratorV2.generateSimpleCustomImage"
                }
            }]/**SCHEMA_DIFF*/,
        methods: {
            getReadImageUrl: function() {
                /* Custom implementation. */
                console.log("Contains custom logic");
                /* Call the mixin method. */
                this.mixins.ContentImageMixin.getImageUrl.apply(this, arguments);
            }
        },
        rules: {}
    };
});
```

Click [*Save*] on the Designer toolbar.

Method declaration example

Medium

Example. Add the [*Email*] column validation logic to the logic of the `setValidationConfig` method located in the `Terrasoft.configuration.BaseSchemaViewModel` class.

Protected method example

```
methods: {
    /* Method name. */
    setValidationConfig: function() {
        /* Call the setValidationConfig method of the parent schema. */
        this.callParent(arguments);
        /* Set up validation for the [Email] column. */
        this.addColumnValidator("Email", EmailHelper.getEmailValidator);
    }
}
```

© 2023 Creatio. All rights reserved.

New method example

```
methods: {
    /* Method name. */
    getBlankSlateHeaderCaption: function() {
        /* Get the value of the MasterColumnInfo column. */
        var masterColumnInfo = this.get("MasterColumnInfo");
        /* Return the result value of the method. */
        return masterColumnInfo ? masterColumnInfo.caption : "";
    },
    /* Method name. */
    getBlankSlateIcon: function() {
        /* Return the result value of the method. */
        return this.Terrasoft.ImageUrlBuilder.getUrl(this.get("Resources.Images.BlankSlateIcon")
    }
}
```

Array of modifications usage example

Medium

```
diff: /**SCHEMA_DIFF*/[
  {
      "operation": "insert",
      "name": "CardContentWrapper",
      "values": {
          "id": "CardContentWrapper",
          "itemType": Terrasoft.ViewItemType.CONTAINER,
          "wrapClass": "card-content-container"],
          "items": []
      }
  },
  {
      "operation": "insert",
      "name": "CardContentContainer",
      "parentName": "CardContentWrapper",
      "propertyName": "items",
      "values": {
          "itemType": Terrasoft.ViewItemType.CONTAINER,
          "items": []
      }
  },
  {
```

© 2023 Creatio. All rights reserved.

```
        "operation": "insert",
        "name": "HeaderContainer",
        "parentName": "CardContentContainer",
        "propertyName": "items",
        "values": {
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            "wrapClass": ["header-container-margin-bottom"],
            "items": []
        }
    },
    {
        "operation": "insert",
        "name": "Header",
        "parentName": "HeaderContainer",
        "propertyName": "items",
        "values": {
            "itemType": Terrasoft.ViewItemType.GRID_LAYOUT,
            "items": [],
            "collapseEmptyRow": true
        }
    }
]/**SCHEMA_DIFF*/
```

Example of using the alias mechanism for repeated schema replacement

Medium

The `diff` array of modifications has an initial "`Name`" element with a set of properties. The element is located in the `Header` container. This schema is replaced several times. The "`Name`" element is modified and moved freely.

diff property of the base schema

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Insert operation. */
        "operation": "insert",
        /* The name of the parent element into which to insert the element. */
        "parentName": "Header",
        /* The name of the parent element property on which to operate. */
        "propertyName": "items",
        /* The name of the element. */
        "name": "Name",
        /* The object of element property values. */
        "values": {
```

© 2023 Creatio. All rights reserved.

```
            /* Layout. */
            "layout": {
                /* Column number. */
                "column": 0,
                /* The row number. */
                "row": 1,
                /* The number of combined columns. */
                "colSpan": 24
            }
        }
    }
] /**SCHEMA_DIFF*/
```

diff property after the first replacement of the base schema

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* The operation that combines the properties of two elements. */
        "operation": "merge",
        "name": "Name",
        "values": {
            "layout": {
                "column": 0,
                /* The row number. The element has been moved. */
                "row": 8,
                "colSpan": 24
            }
        }
    }
] /**SCHEMA_DIFF*/
```

diff property after the second replacement of the base schema

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Move operation. */
        "operation": "move",
        "name": "Name",
        /* The name of the parent element where the move operation is done. */
        "parentName": "SomeContainer"
    }
] /**SCHEMA_DIFF*/
```

© 2023 Creatio. All rights reserved.

In the new version, the element named "Name" has been moved from the `SomeContainer` element to the `ProfileContainer` element and must remain there despite the client customization. To enforce this, the element gets a new name "NewName" and the `alias` configuration object is added to it.

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Insert operation. */
        "operation": "insert",
        /* The name of the parent element into which to insert the element. */
        "parentName": "ProfileContainer",
        /* The name of the parent element property on which to operate. */
        "propertyName": "items",
        /* New name. */
        "name": "NewName",
        /* The object of element property values . */
        "values": {
            /* Bind to the property or function value. */
            "bindTo": "Name",
            /* Layout. */
            "layout": {
                /* Column number. */
                "column": 0,
                /* The row number. */
                "row": 0,
                /* The number of combined columns. */
                "colSpan": 12
            }
        },

        /* alias configuration object. */
        "alias": {
            /* The old name of the element. */
            "name": "Name",
            /* An array of custom replacing properties to ignore. */
            "excludeProperties": "layout" ],
            /* An array of custom replacing operations to ignore. */
            "excludeOperations": [ "remove", "move" ]
        }
    }
] /**SCHEMA_DIFF*/
```

The new element now has `alias`. The parent element changed, as is its location on the record page. The `excludeProperties` property stores a set of properties that will be ignored when the delta is applied, while `excludeOperations` stores a set of operations that will not be applied to this element from the replacements.

In this example, the `layout` properties of all "Name" descendants are excluded, and the `remove` and `move` operations are also prohibited. This means that the "NewName" element will contain only the root `layout` property

© 2023 Creatio. All rights reserved.

and all properties of the "Name" element from the replacements except `Layout`. The same applies to operations.

Result for the builder of the `diff` array of modifications

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Insert operation. */
        "operation": "insert",
        /* The name of the parent element into which to insert the element. */
        "parentName": "ProfileContainer",
        /* The name of parent element property on which to operate. */
        "propertyName": "items",
        /* New name. */
        "name": "NewName",
        /* The object of element property values. */
        "values": {
            /* Bind to the property or function value. */
            "bindTo": "Name",
            /* Layout. */
            "layout": {
                /* Column number. */
                "column": 0,
                /* The row number. */
                "row": 0,
                /*/ The number of combined columns. */
                "colSpan": 12
            },
        }
    },
] /**SCHEMA_DIFF*/
```

attributes property JS

Beginner

The `attributes` property of the client schema contains a configuration object with its properties.

Primary properties

`dataValueType`

The attribute data type. Creatio will use it when generating the view. The `Terrasoft.DataValueType` enumeration represents the available data types.

Available values (`DataValueType`)

© 2023 Creatio. All rights reserved.

GUID	0
TEXT	1
INTEGER	4
FLOAT	5
MONEY	6
DATE_TIME	7
DATE	8
TIME	9
LOOKUP	10
ENUM	11
BOOLEAN	12
BLOB	13
IMAGE	14
CUSTOM_OBJECT	15
IMAGELOOKUP	16
COLLECTION	17
COLOR	18
LOCALIZABLE_STRING	19
ENTITY	20
ENTITY_COLLECTION	21
ENTITY_COLUMN_MAPPING_COLLECTION	22
HASH_TEXT	23
SECURE_TEXT	24

© 2023 Creatio. All rights reserved.

FILE	25
MAPPING	26
SHORT_TEXT	27
MEDIUM_TEXT	28
MAXSIZE_TEXT	29
LONG_TEXT	30
FLOAT1	31
FLOAT2	32
FLOAT3	33
FLOAT4	34
LOCALIZABLE_PARAMETER_VALUES_LIST	35
METADATA_TEXT	36
STAGE_INDICATOR	37

`type`

Column type. An optional parameter `BaseViewModel` uses internally. The `Terrasoft.ViewModelColumnType` enumeration represents the available column types.

Available values (`ViewModelColumnType`)

ENTITY_COLUMN	0
CALCULATED_COLUMN	1
VIRTUAL_COLUMN	2
RESOURCE_COLUMN	3

`value`

Attribute value. Creatio sets the view model value to this parameter when the view model is created. The `value` attribute accepts numeric, string, and boolean values. If the attribute type implies the use of a lookup

© 2023 Creatio. All rights reserved.

type value (array, object, collection, etc.), initialize its initial value using a method.

Use example

Example that uses basic attribute properties

```
attributes: {
    /* Attribute name. */
    "NameAttribute": {
        /* Data type. */
        "dataValueType": this.Terrasoft.DataValueType.TEXT,
        /* Column type. */
        "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
        /* The default value. */
        "value": "NameValue"
    }
}
```

Additional properties

`caption`

Attribute title.

`isRequired`

The flag that marks the attribute as required.

`dependencies`

Dependency on another attribute of the model. For example, set an attribute based on the value of another attribute. Use the property to create calculated fields.

`lookupListConfig`

The property that manages the lookup field properties. Learn more about using this parameter in a separate article: Filter the lookup field. This is a configuration object that can contain optional properties.

Optional properties

© 2023 Creatio. All rights reserved.

`columns`	An array of column names to add to a request in addition to the [*Id*] column and the primary display column.
`orders`	An array of configuration objects that determine the data sorting.
`filter`	The method that returns an instance of the `Terrasoft.BaseFilter` class or its descendant that will be applied to the request. Cannot be used combined with the `filters` property.
`filters`	An array of filters (methods that return collections of the `Terrasoft.FilterGroup` class). Cannot be used combined with the `filter` property.

Use example

Example that uses additional attribute properties

```
attributes: {
    /* Attribute name. */
    "Client": {
        /* Attribute title. */
        "caption": { "bindTo": "Resources.Strings.Client" },
        /* The attribute is required. */
        "isRequired": true
    },

    /* Attribute name. */
    "ResponsibleDepartment": {
        lookupListConfig: {
            /* Additional columns. */
            columns: "SalesDirector",
            /* Sorting column. */
            orders: [ { columnPath: "FromBaseCurrency" } ],
            /* Filter definition function. */
            filter: function()
            {
                /* Return a filter by the [Type] column, which equals the Competitor consta
                return this.Terrasoft.createColumnFilterWithParameter(
                this.Terrasoft.ComparisonType.EQUAL,
                "Type",
                ConfigurationConstants.AccountType.Competitor);
            }
        }
    },
    /* Attribute name. */
    "Probability": {
        /* Define the column dependency. */
```

© 2023 Creatio. All rights reserved.

```
        "dependencies": [
        {
            /* Depends on the Stage] column. */
            "columns": [ "Stage" ],
            /* The name of the [Stage] column's change handler method.
            The setProbabilityByStage() method is defined in the methods property of the sc
            "methodName": "setProbabilityByStage"
        }
        ]
    }
},
methods: {
    /* [Stage] column's change handler method. */
    setProbabilityByStage: function()
    {
        /* Get the value of the [Stage] column. */
        var stage = this.get("Stage");
        /* Condition for changing the [Probability] column. */
        if (stage.value && stage.value ===
        ConfigurationConstants.Opportunity.Stage.RejectedByUs)
        {
            /* Set the value of the [Probability] column. */
            this.set("Probability", 0);
        }
    }
}
```

messages property JS

Medium

The `messages` property of the client schema contains a configuration object with its properties.

Properties

`mode`

Message mode. The `Terrasoft.MessageMode` enumeration represents the available modes.

Available values (`MessageMode`)

PTP	Address.
BROADCAST	Broadcasting.

© 2023 Creatio. All rights reserved.

`direction`

Message direction. The `Terrasoft.MessageDirectionType` enumeration represents the available modes.

Available values (`MessageDirectionType`)

`PUBLISH`	Publishing.
`SUBSCRIBE`	Subscription.
`BIDIRECTIONAL`	Bidirectional.

rules and businessRules properties JS

Beginner

The `rules` and `businessRules` properties of the client schema contain a configuration object with its own properties.

Primary properties

`ruleType`

Rule type. Defined by the `BusinessRuleModule.enums.RuleType` enumeration value.

Available values (`BusinessRuleModule.enums.RuleType`)

`BINDPARAMETER`	Business rule type. Use this rule type to link properties of a column to values of different parameters. For example, set up the visibility of a column or enable a column depending on the value of another column.
`FILTRATION`	Business rule type. Use the FILTRATION rule to set up filtering of values in view model columns. For example, filter a `LOOKUP` column depending on the current status of a page.

`property`

Use for the `BINDPARAMETER` business rule type. Control property. Set by the `BusinessRuleModule.enums.Property` enumeration value.

Available values (`BusinessRuleModule.enums.Property`)

© 2023 Creatio. All rights reserved.

`VISIBLE`	Whether visible.
`ENABLED`	Whether available.
`REQUIRED`	Whether required.
`READONLY`	Whether read-only.

`conditions`

Use for the `BINDPARAMETER` business rule type. Condition array for rule application. Each condition is a configuration object.

Properties of the configuration object

© 2023 Creatio. All rights reserved.

<table>
<tr><td><code>leftExpression</code></td><td>Expression of the left side of the condition. Represented by a configuration object.
Properties of the configuration object

<code>type</code>

The expression type. Set by the <code>BusinessRuleModule.enums.ValueType</code> enumeration value.

Available values (<code>BusinessRuleModule.enums.ValueType</code>)
<table><tr><td><code>CONSTANT</code></td><td>A constant.</td></tr><tr><td><code>ATTRIBUTE</code></td><td>The value of the view model column.</td></tr><tr><td><code>SYSSETTING</code></td><td>System setting.</td></tr><tr><td><code>SYSVALUE</code></td><td>A system value. The list element of the <code>Terrasoft.core.enums.SystemValueType</code> system values.</td></tr></table>
<code>attribute</code>

Name of the model column.

<code>attributePath</code>

Meta-path to the lookup schema column

<code>value</code>

Comparison value.</td></tr>
<tr><td><code>comparisonType</code></td><td>Type of comparison. Set by the <code>Terrasoft.core.enums.ComparisonType</code> enumeration value.</td></tr>
<tr><td><code>rightExpression</code></td><td>Expression of the right side of the condition. Similar to <code>leftExpression</code>.</td></tr>
</table>

`logical`

Use for the `BINDPARAMETER` business rule type. The logical operation that combines the conditions from the conditions property. Set by the `Terrasoft.LogicalOperatorType` enumeration value.

© 2023 Creatio. All rights reserved.

`autocomplete`

Use for the `FILTRATION` business rule type. Reverse filtering flag. Can be `true` or `false`.

`autoClean`

Use for the `FILTRATION` business rule type. The flag that enables automated value cleanup when the column by which to filter changes. Can be `true` or `false`.

`baseAttributePatch`

Use for the `FILTRATION` business rule type. Meta-path to the lookup schema column that will be used for filtering. Apply the feedback principle when building the column path, similar to `EntitySchemaQuery`. Generate the path relative to the schema to which the model column links.

`comparisonType`

Use for the `FILTRATION` business rule type. Type of comparison operation. Set by the `Terrasoft.ComparisonType` enumeration value.

`type`

Use for the `FILTRATION` business rule type. The value type for comparison `baseAttributePatch`. Set by the `BusinessRuleModule.enums.ValueType` enumeration value.

`attribute`

Use for the `FILTRATION` business rule type. The name of the view model column. Describe this property if the `ATTRIBUTE` value `type` is indicated.

`attributePath`

Use for the `FILTRATION` business rule type. Meta-path to the object schema column. Apply the feedback principle when building the column path, similar to `EntitySchemaQuery`. Generate the path relative to the schema to which the model column link.

`value`

Use for the `FILTRATION` business rule type. Filtration value. Describe this property if the `ATTRIBUTE` value `type` is indicated.

Additional properties

© 2023 Creatio. All rights reserved.

Use additional properties only for the `businessRules` property.

`uId.`

Unique rule ID. The "`GUID`" type value.

`enabled`

Enabling flag. Can be `true` or `false`.

`removed`

The flag that indicates whether the rule is removed. Can be `true` or `false`.

`invalid`

The flag that indicates whether the rule is valid. Can be `true` or `false`.

Use examples

Example of a BINDPARAMETER business rule created by the Wizard

```
define("SomePage", [], function() {
    return {
        /* ... */
        businessRules: /**SCHEMA_BUSINESS_RULES*/{
        /* A set of rules for the Type column of the view model. */
        "Type": {
            /* The rule code the Wizard generates. */
            "ca246daa-6634-4416-ae8b-2c24ea61d1f0": {
                /* Unique rule ID. */
                "uId": "ca246daa-6634-4416-ae8b-2c24ea61d1f0",
                /* Enabling flag. */
                "enabled": true,
                /* The flag that indicates whether the rule is removed. */
                "removed": false,
                /* The checkbox that indicates whether the rule is valid. */
                "invalid": false,
                /* Rule type. */
                "ruleType": 0,
                /* The code for the property that controls the rule. */
                "property": 0,
                /* A logical relationship between several rule conditions. */
                "logical": 0,
                /* An array of conditions that trigger the rule.
```

© 2023 Creatio. All rights reserved.

```
               Compare the Account.PrimaryContact.Type value to the Type column value. */
                "conditions": [
                    {
                    /* Type of comparison operation. */
                    "comparisonType": 3,
                    /* Expression of the left side of the condition. */
                    "leftExpression": {
                        /* The expression type is a column (attribute) of a view model. */
                        "type": 1,
                        /* The name of the view model column. */
                        "attribute": "Account",
                        /* Path to the column in the Account lookup schema, whose value to
                        "attributePath": "PrimaryContact.Type"
                    },
                    /* Expression of the right side of the condition. */
                    "rightExpression": {
                        /* The expression type is a column (attribute) of a view model. */
                        "type": 1,
                        /* The name of the view model column. */
                        "attribute": "Type"
                    }
                    }
                ]
            }
        }
        }/**SCHEMA_BUSINESS_RULES*/
        /* ... */
    };
});
```

Example of a FILTRATION business rule created by the Wizard

```
define("SomePage", [], function() {
    return {
        /* ... */
        businessRules: /**SCHEMA_BUSINESS_RULES*/{
        /* A set of rules for the Type column of the view model. */
        "Account": {
            /* The rule code the Wizard generates. */
            "a78b898c-c999-437f-9102-34c85779340d": {
                /* Unique rule ID. */
                "uId": "a78b898c-c999-437f-9102-34c85779340d",
                /* Enabling flag. */
                "enabled": true,
                /* The flag that indicates whether the rule is removed. */
                "removed": false,
                /* The flag that indicates whether the rule is valid. */
```

© 2023 Creatio. All rights reserved.

```
                "invalid": false,
                /* Rule type. */
                "ruleType": 1,
                /* Path to the column for filtering in the Account lookup schema reference
                "baseAttributePatch": "PrimaryContact.Type",
                /* The comparison type in the filter. */
                "comparisonType": 3,
                /* Expression type is a column (attribute) of a view model. */
                "type": 1,
                /* Name of the view model column, by which to filter the records. */
                "attribute": "Type"
            }
        }
        }/**SCHEMA_BUSINESS_RULES*/
        /* ... */
    };
});
```

diff property JS

The `diff` property of the client schema contains an array of configuration objects with their properties.

Properties

`operation`

Operation on elements.

Available values

`set`	Setsthe schema element to the value determined by the `values` parameter.
`merge`	Merges the values from the parent, replaced, and replacing schemas. The properties from the `values` parameter of the last descendant override the other values.
`remove`	Deletes the element from the schema.
`move`	Moves the element to a different parent element.
`insert`	Adds the element to the schema.

© 2023 Creatio. All rights reserved.

`name`

The name of the schema element on which the operation is run.

`parentName`

The name of the parent schema element where to place the element during the `insert` operation, or to which to move the element during the `move` operation.

`propertyName`

The name of the parent element parameter for the `insert` operation. Also used in the `remove` operation when only certain element parameters must be removed rather than the entire element.

`index`

The index to which to move or add the parameter. Use the parameter with the `insert` and `move` operations. If the parameter is not specified, then insert is the last element of the array.

`values`

An object whose properties to set or combine with the properties of the schema element. Use with `set`, `merge` and `insert` operations.

The `Terrasoft.ViewItemType` enumeration represents the set of basic elements that can be displayed on the page

Available values (`ViewItemType`)

GRID_LAYOUT	0	A grid element that includes the placement of other controls.
TAB_PANEL	1	A set of tabs.
DETAIL	2	Detail
MODEL_ITEM	3	View model element.
MODULE	4	Module.
BUTTON	5	Button.
LABEL	6	Caption.
CONTAINER	7	Containers.
MENU	8	Drop-down list.

© 2023 Creatio. All rights reserved.

MENU_ITEM	9	Drop-down list element.
MENU_SEPARATOR	10	Drop-down list separator.
SECTION_VIEWS	11	Section views.
SECTION_VIEW	12	Section view.
GRID	13	List.
SCHEDULE_EDIT	14	Scheduler.
CONTROL_GROUP	15	Element group.
RADIO_GROUP	16	Switcher group.
DESIGN_VIEW	17	Configurable view.
COLOR_BUTTON	18	Color.
IMAGE_TAB_PANEL	19	A set of tabs with icons.
HYPERLINK	20	Hyperlink.
INFORMATION_BUTTON	21	Info button that has a tooltip.
TIP	22	Tooltip.
COMPONENT	23	Component.
PROGRESS_BAR	30	Indicator.

`alias`

Configuration object.

`alias` object properties

© 2023 Creatio. All rights reserved.

`name`	The name of the element to which the new element is connected. Creatio will use this name to locate the elements in the replaced schemas and to connect the elements with the new element. The `name` property of the `diff` revision array element cannot equal the `alias.name` property.
`excludeProperties`	An property array of the `values` object of the element from the `diff` modification array. The properties will not be applied when building `diff`.
`excludeOperations`	An array of operations that must not be applied to this element when building the `diff` array of modifications.

Use example

Example that uses the `alias` object

```
/* diff array. */
diff: /**SCHEMA_DIFF*/ [
  {
    /* The operation to perform on the element. */
    "operation": "insert",
    /* New name. */
    "name": "NewElementName",
    /* Element value. */
    "values": {
        /* ... */
    },
    /* alias configuration object. */
    "alias": {
        /* The previous name of the element. */
        "name": "OldElementName",
        /* An array of excluded properties. */
        "excludeProperties": "layout", "visible", "bindTo" ],
        /* An array of ignored operations. */
        "excludeOperations": [ "remove", "move", "merge" ]
    }
  },
  /* ... */
]
```

© 2023 Creatio. All rights reserved.