Creatio Academy

Front-end development Freedom UI

Creatio front-end architecture

Version 8.0

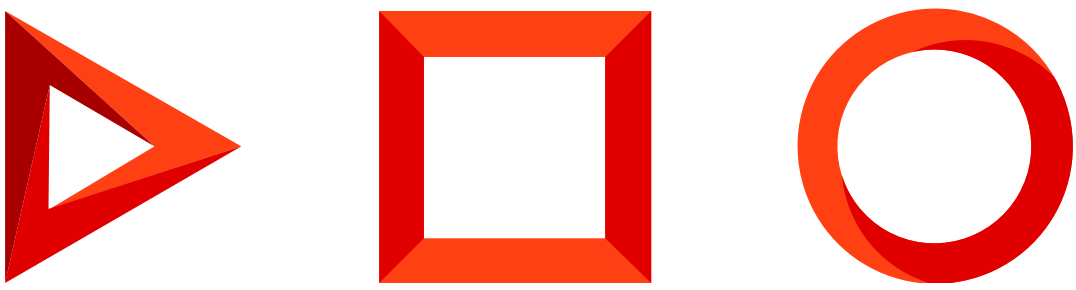


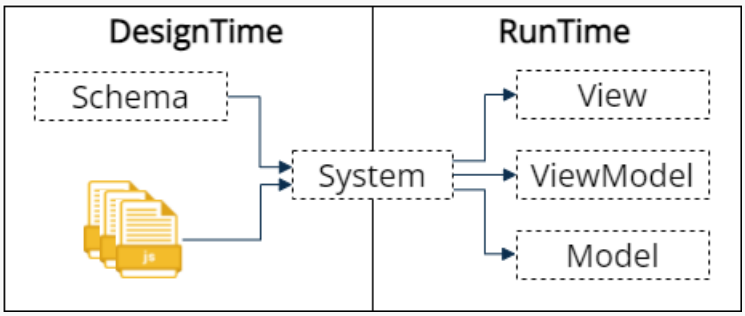
Table of Contents

Creatio front-end architecture

Beginner

The **software platform** is an environment required for Creatio development (`DesignTime` mode) and runtime (`RunTime` mode).

View the interaction pattern of structural items in the Creatio platform in the figure below.



Structural mode items of the Creatio platform interact via the `System` layer.

`DesignTime` mode

The **purpose** of the `DesignTime` mode is Creatio development, extension, and customization.

The `DesignTime` mode of the Creatio platform includes the following **structural items**:

- `Schema` metadata layer. Contains a set of client schemas. Learn more about client schemas in a separate article: Client schema.
- Precompiled JavaScript code.

`RunTime` mode

The `RunTime` mode of the Creatio platform includes the following **structural items**:

- The `View` layer for data visualization.
- The `ViewModel` layer for interaction between the `View` and `Model` layers.
- The `Model` data layer.

`View` layer

The `View` layer handles data visualization. Represented as a set of visual components of the `DesignTime` mode.

Components of various types implement the `View` layer. For example, these can be components that store nested components (e. g., `GridContainerComponent`), components that display data (e. g., `LabelComponent`),

components that interact with the user (e. g., `InputComponent`), etc.

You can use components in the `viewConfigDiff` section of the client schema. If you use components in client schemas, Creatio generates components based on the schema metadata. This lets you customize the view using Creatio low-code/no-code tools.

`ViewModel` layer

`ViewModel` layer handles the business logic of the `View` and `Model` layer interaction. Represented by the `ViewModel` type that encapsulates the attribute management logic: data initialization, binding to visual component sections, change tracking. Creatio does not support custom section `view model` types.

The `ViewModel` layer lets you implement the following business logic **types**:

- validators
- converters
- query handlers

Learn more in a separate article: Client schema.

The following attribute **types** implement the `ViewModel` layer:

- simple type attribute (`string`, `number`, `boolean`)
- attribute that contains nested `view model` instances
- resource attribute (`readonly`)

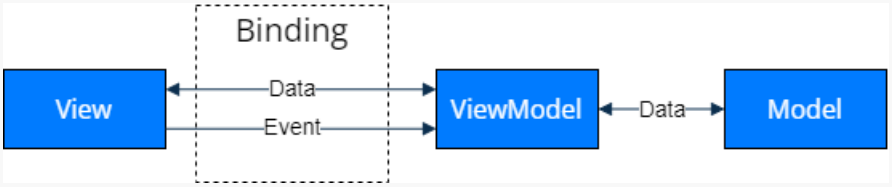
Creatio `ViewModel` layer supports the following life cycle **stages**:

- Instance initialization (`crt.HandleViewModelInitRequest` handler).
- Addition of a `View` model to a container using the `attach` operation (`crt.HandleViewModelResumeRequest` handler). Creatio calls the handler when you open a module. Available in Creatio 8.0.6 Atlas and later.
- Attribute value change (`crt.HandleViewModelAttributeChangeRequest` handler).
- Removal of the `View` model of the current module from a container using the `detach` operation (`crt.HandleViewModelPauseRequest` handler). Creatio calls the handler when you switch to another module. Available in Creatio 8.0.6 Atlas and later.
- Instance destruction (`crt.HandleViewModelDestroyRequest` handler). At this stage, execute only synchronous code that destroys resources accumulated as part of runtime.

Creatio calls `HandleViewModelResumeRequest` and `HandleViewModelPauseRequest` handlers if the current module has the `viewModel` parameter. The handlers are called in the following order: `HandleViewModelResumeRequest` → `HandleViewModelPauseRequest`. Creatio does not call handlers if you open a Freedom UI page and switch to another module before executing the `attach` operation.

To ensure data visualization in Creatio UI and synchronization of this data operate as intended, bind the `View` layer to `ViewModel`.

View the structure of `View` to `ViewModel` binding in the figure below.



Creatio provides the following binding **types**:

- one way binding to attribute
- binding to resource attribute
- retrieval of the `CrtControl` instance

View an example that uses various binding types below.

Example that uses bindings

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        "FirstName": {},
        "Visible": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "insert",
        "name": "UsrSickDaysLeft",
        type: "crt.Input",
        /* Retrieve a CrtControl instance. */
        control: "$FirstName",
        /* binding to resource attribute */
        title: "$Resources.Strings.Title",
        /* one way binding to attribute */
        visible: "$Visible"
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

You can expand the binding mechanism with the needed macro mechanism. For example, use the macro mechanism to bind nested properties of the object to `ViewModel` resources. A **macro** is an item that replaces part of `view config` with `ViewModel` resources. Unlike a binding, a macro is only triggered once and not synchronized upon further `view model` changes. Creatio 8 Atlas implements only the `#ResourceString#` macro that implements resource string management.

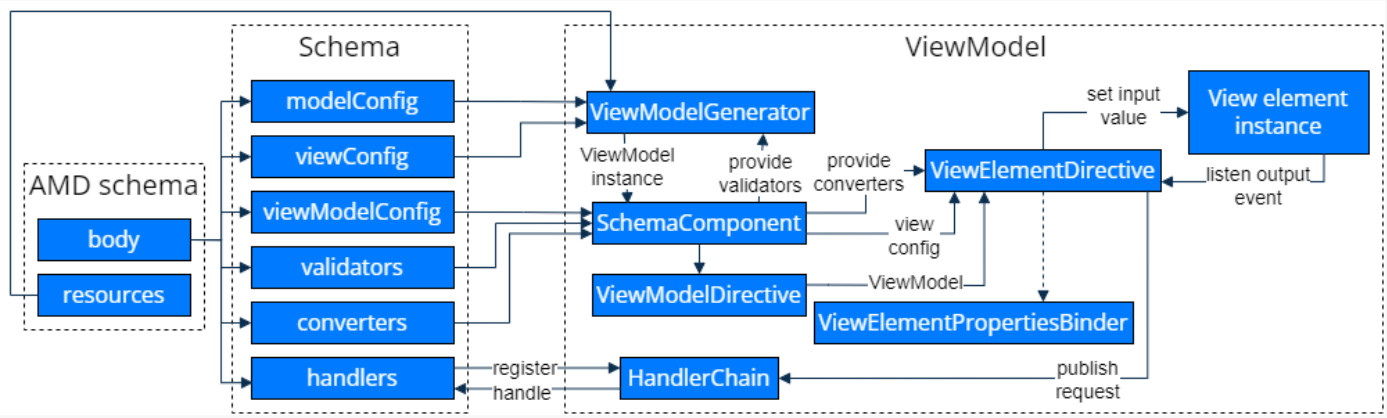
View an example that inserts the value from resources into the `caption` property of the `Header` element below.

Example that uses the `#ResourceString#` macro

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...
    {
        "operation": "insert",
        "name": "Header",
        "values": {
            "type": "crt.Label",
            "caption": "#ResourceString(Header)#",
        },
    ...
    },
    ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/
```

Since custom `ViewModel` instances are not supported, describe the business logic in Creatio 8 Atlas using individual query handlers. You can chain handlers and define the needed fetch time of the handler.

View Creatio workflow in the figure below.

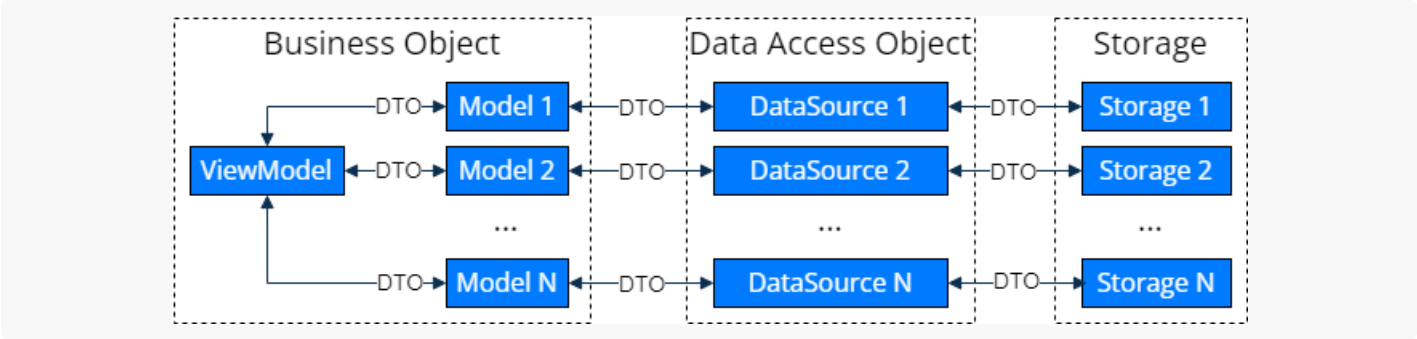


`Model` layer

The `Model` layer handles data management. The layer lets you manage data sources and schema of their data, as well as execute data operations (load, save, delete, sort, etc.). Creatio 8 Atlas includes the `EntityDataSource` data source type that lets you manage data of `Entity` objects. The set of data sources is closed for expansion in Creatio 8 Atlas. This feature will be available in the future. he `ViewModel` layer uses the `Model` layer functionality to supply the `View` layer with data.

Creatio uses the **Data Access Object** (DAO) pattern to store data. Learn more about the DAO pattern in Wikipedia.

View the DAO workflow in Creatio in the figure below.



Items of the `Data Access Object` group solve the following **problems**:

- Ensure the execution of CRUD operations.
- Provide data operation (create, update, delete) permissions.
- Provide the data structure (`DataSchema`).

Creatio 8 Atlas implements `EntityDataSource` that manages the Creatio database.

Initialize the `Model` with `viewModel` based on the workflow below.