WHITESTEIN
Technologies

Living Systems® Process Suite

# Scaffolding Library

## Living Systems Process Suite Documentation

3.6
Mon Nov 1 2021

Whitestein Technologies AG | Hinterbergstrasse 20 | CH-6330 Cham
Tel +41 44-256-5000 | Fax +41 44-256-5001 | http://www.whitestein.com

# Contents

# Chapter 1

# Main Page

To prototype CRUD components in your forms quickly, use the scaffolding framework provided by the modules of the Scaffolding Library. They enable you to create form components that can create, read, update, and delete objects of a particular type, mainly shared record type.

> **Important:** The Scaffolding Library resources are not optimized and hence not intended for production environment.

The library contains two modules:

- scaffolding_forms for use with `forms created with the forms module`
- scaffolding_ui for use with `forms created with the ui module`

# Chapter 2

# Creating CRUD Table in UI Forms

To create a CRUD table over a collection of objects, use the Generic View component provided by the Scaffolding Library: the component automatically generates the table

Note that scaffolding on UI forms is primarily intended for records and might not work if its objects are of other data type or of different types. From the table you will be able to create a new instances of the records, and delete and modify the existing ones.

For example, if you have a record Book with the fields `isbn` and `title`, and you define a list of book instances as the content of the Generic View component, the rendered table will have the columns isbn and title.

To create a CRUD component in a ui form, do the following:

1. Make sure our project contains the Scaffolding Library.
2. Import the *scaffolding_ui* module into your module.
3. In your form definition, insert the *Generic View* component available under Custom Components.

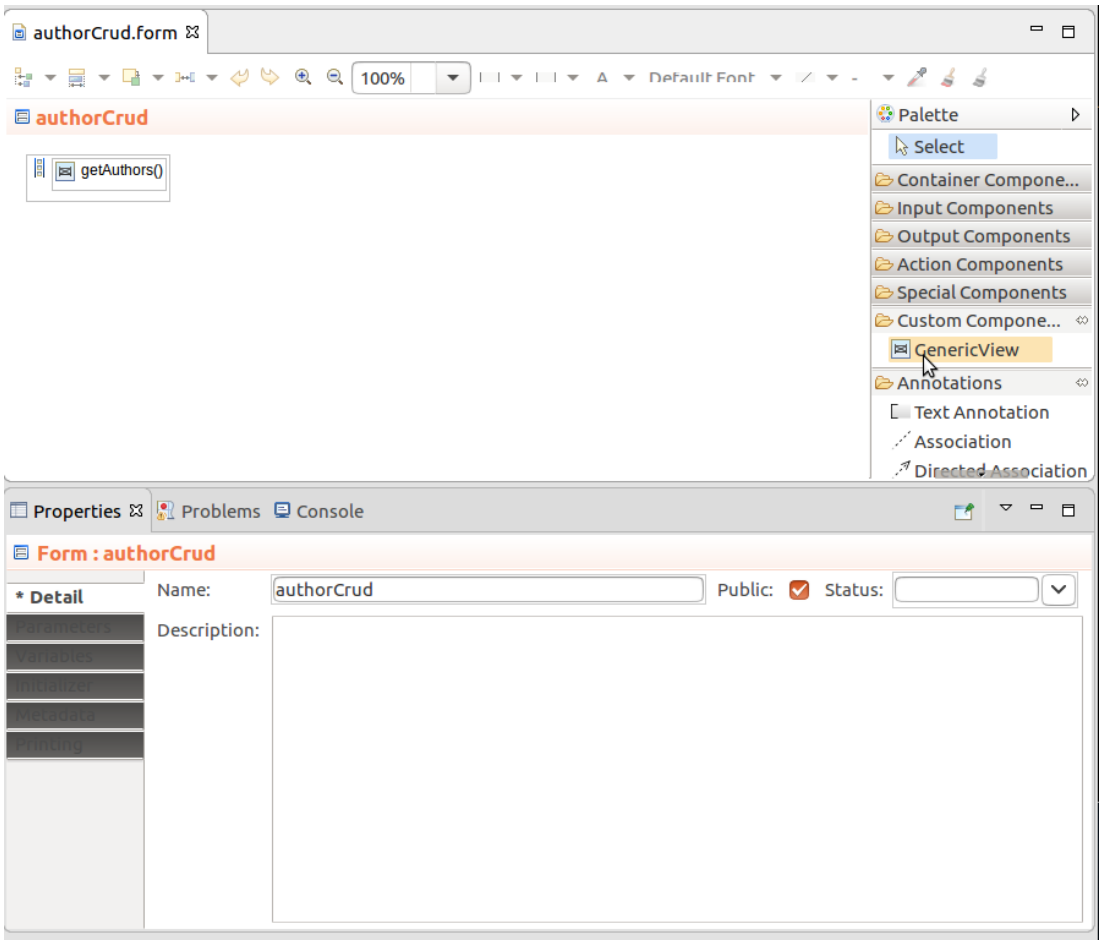


**Figure 2.1 Generic view in palette**

4. Define the properties of the *Generic View* component:

   **Object** List of Objects that are the content of the form component

   Each object is represented by a table row.

   For object of a simple type, their value is rendered in a sole column; for objects of a record type, the value of each field is renderer in its own column; Note that this can be overridden by the *record options* property.

   **Editable** If true, the table is editable: if the table contains different types of objects, the table remains read-only.

   **Object Type** If the Object expression returns an empty Collection, the table is considered to operate over this data type, hence if the table is editable and you click *Create new instance*, the table creates a new instance of this type).

   **Recursion Depth** The relationship depth from the table record that is displayed directly in the component of the top data type: the related data types are displayed as sub-node components. They can be expanded directly in the parent component. Otherwise, the data is displayed on a new page and the depth is indicated only in the breadcrumb.

   **Submit Button** The component rendered instead of the default Submit button

   This expression has no impact on the **OK & Persist** button, which is displayed when creating a new instance in a Collection of shared Records from the Generic View.

   **Record options** Custom record rendering

   If defined, the table renders does not use the generic CRUD table but the form defined in this map to render the record data whenever it appears in the Generic View.

   [

```
        Author -> new RecordOptions(
          linkDisplayName -> {o:Object-> cast(o, Author).name},
          recordCustomForm -> {o:Object -> new OutputText(content ->{->"anonymous"})}
        ),
        Book -> new RecordOptions(
          propertyVisibilityDefault -> false,
          propertyOptions -> [
            Book.name -> new PropertyOptions(
              propertyVisibility -> true
            ),
            Book.authors -> new PropertyOptions(
              propertyVisibility -> true
            )
          ]
        )
      ]
```

## 2.1 Recursion Depth

When the Generic view has records as its content and the record has a relationship to another record, the rendered CRUD table contains a column with a link to the related record. When you click the link, the data of the related record is displayed in its own component. You can navigate back with a breadcrumb.

However, you can display such related objects as part of the CRUD component: define the Recursive property on your Generic View as an Integer that represent the number of relationships to navigate through: The reachable record instances are rendered as a nested component that can be expanded.

> **Note:** The component immediately loads the data within the reach of one relationship so that when requested the data is already available.

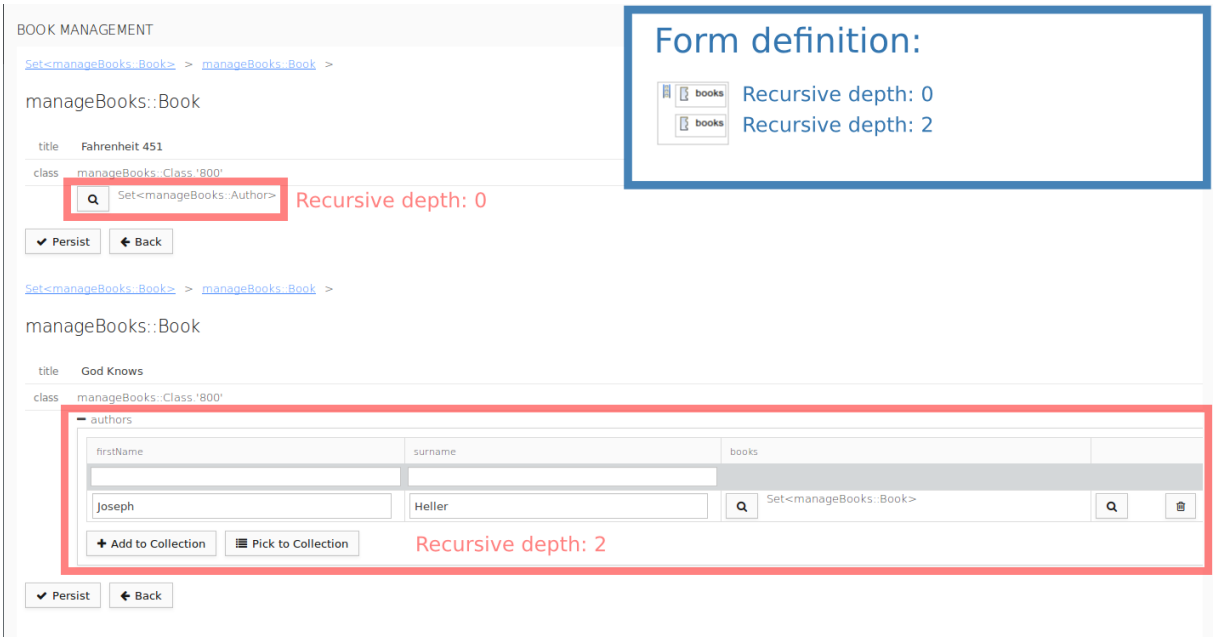


**Figure 2.2 Document with Generic Views. The top Generic View has the Recursive depth property set to 0 while the bottom Generic View has the Recursive depth set to 2.**

# Chapter 3

# Creating CRUD Components in Forms

CRUD components are implemented in the `scaffolding-forms` module of the Scaffolding Library by the following components:

- **Entity Detail** renders as a form with data of a shared record
- **Entity List** renders as a grid with shared record fields and the option to display their details in a popup as well as the Create button below the table

Note that Scaffolding components work only over data of a shared record type.

## 3.1 Creating a CRUD Component over One Entity

The CRUD component for a single entity form is called the *EntityDetail* component. It provides editable values of a record instance along with any related record instances.

To create a CRUD component over one shared record instance, do the following:

1. Import the `scaffolding-forms` module into your Project and Module:
   (a) In the GO-BPMN Explorer, right-click your Project, click Add Library; in the popup select Select a built-in library and select the Scaffolding Library in the Library drop-down box.
   (b) In the GO-BPMN Explorer, click your module. In the Imports tab of the Properties view, click Add. In the pop-up, select the scaffolding-forms module.
2. Into a form, insert the *Entity Detail* component available under Custom Components.
3. Define the `Record Proxy Set` property of the component. This is the proxy Set that holds the entity displayed in the EntityDetail. For example, you can create a form variable proxySet and initialize it in the form constructor so it creates a proxy of the entity:

```
AuthorCrudDetailForm {
    public AuthorCrudDetailForm(){
      //form variable with the proxy set:
      proxySet := createProxySet(null);
      //form variable with the proxy entity:
      orwellProxy := proxySet.proxy(getAuthorByName("orwell"));
    } ...
```

4. Set the proxy record from the proxy set as the underlying entity with the `setEntity()` call, for example on the Init tab, `c.setEntity(orwellProxy)`.
5. If required, create a component that will call `merge()` on your proxy set to apply any changes.

```
//example click listener on a Submit button
{ e -> proxySet.merge(false); Forms.submit()}
```

## 3.2 Creating a CRUD Component over Multiple Entities

To create a CRUD component with all shared record instances displayed in a tabular manner, do the following:

1. Import the `scaffolding-forms` module into your Project and Module:
   (a) In the GO-BPMN Explorer, right-click your Project, click Add Library; in the popup select Select a built-in library and select the Scaffolding Library in the Library drop-down box.
   (b) In the GO-BPMN Explorer, click your module. In the Imports tab of the Properties view, click Add. In the pop-up, select the scaffolding-forms module.
2. Into your form, insert the *Entity Overview* component available under Custom Components.
3. Define the Entity Type in the properties of the component: This is the shared record type of your data.
4. To allow selecting rows, call `setSelectable(true)` on the component. You can request the current selection with the `getSelection()` call.

To create a new record, click the Create button below the table. If you the component uses a record that has subtypes, they will be able to select the subtype they want to create.

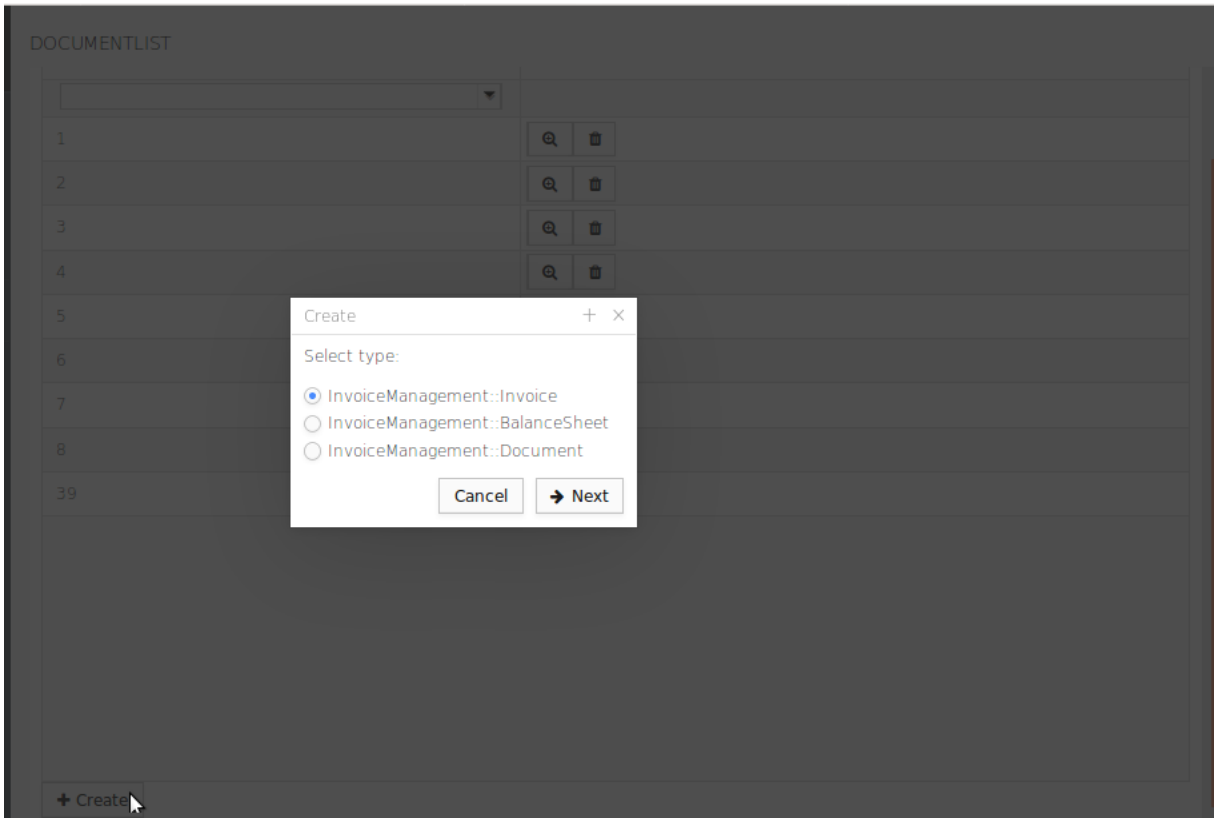


**Figure 3.1 Selecting a subtype for a new shared record**

Note that details of each entity can be displayed in a popup by clicking the **Open Detail** button . If the underlying shared record has a relationship to another record, you can open the details of the related record from the popup.

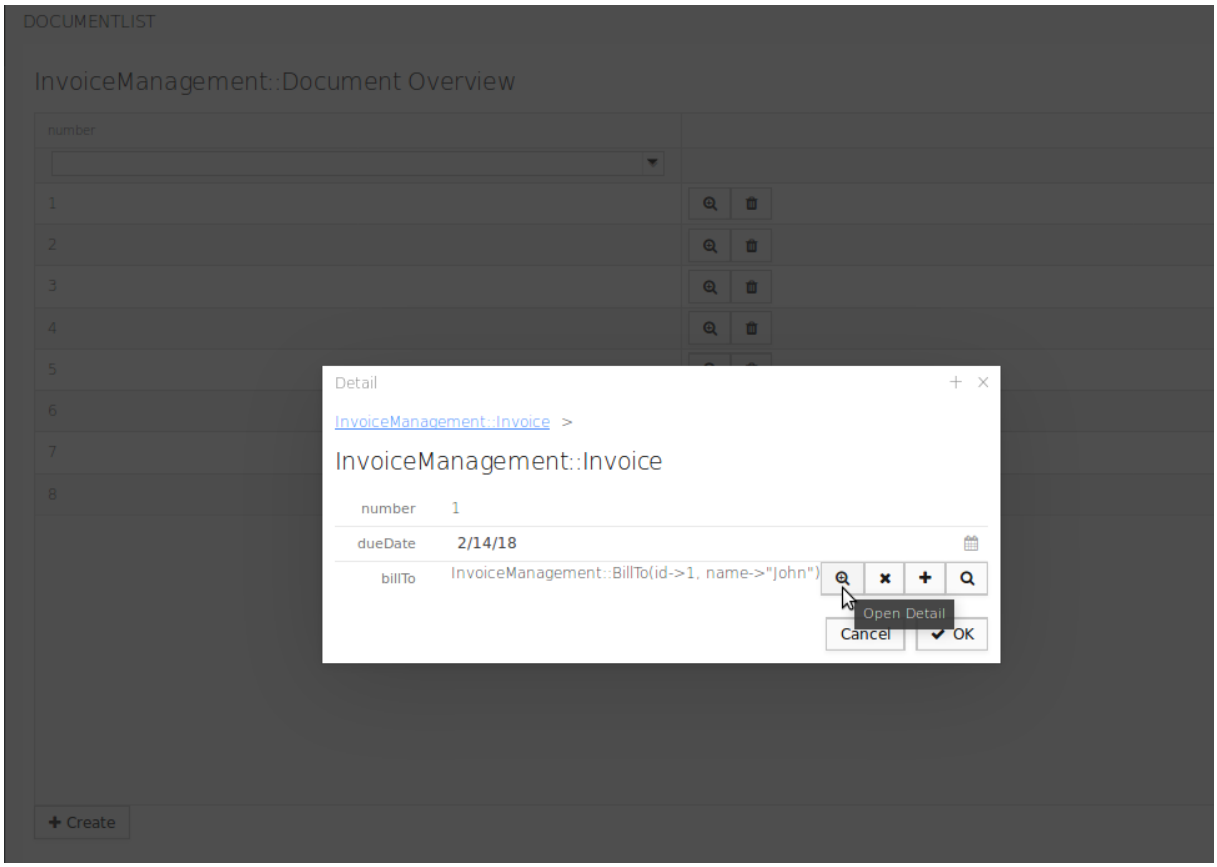


**Figure 3.2 Opening detail of a related record**