

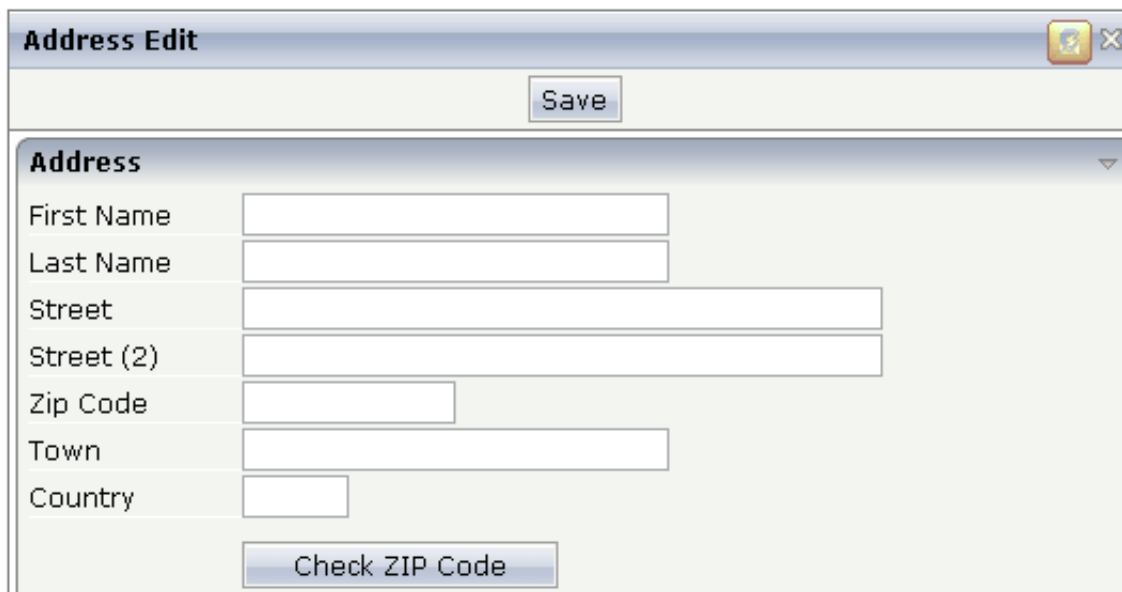
Customized Layout - Concepts

This chapter covers the following topics:

- Overview
 - Dynamic Controls
 - Using Filters
 - Personalization Filter
 - Personalization Scenario Sequence
 - Maintaining Personalization Data
 - Persisting Personalization Data
-

Overview

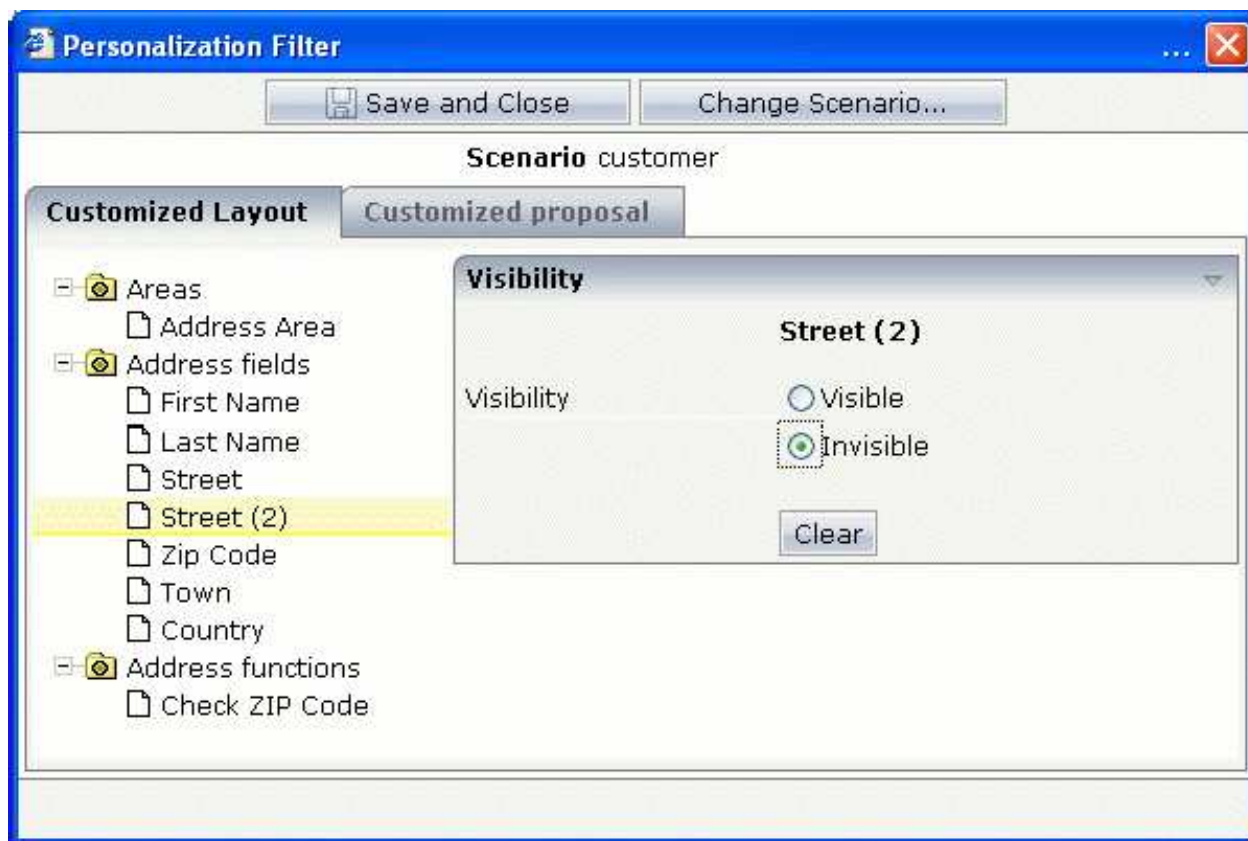
Let us approach the framework from the result side. The following screen is a simple address maintenance:



The screenshot shows a window titled "Address Edit" with a "Save" button at the top. Below the button is a section titled "Address" containing several input fields: "First Name", "Last Name", "Street", "Street (2)", "Zip Code", "Town", and "Country". At the bottom of the form is a "Check ZIP Code" button. In the top right corner of the window, there is a small yellow icon with a lightning bolt and a close button (X).

You see that in the right top corner there is a special icon directly to left of the close icon. This "personalization icon" is not always visible. It is only visible, if a certain parameter is defined inside the session context (e.g. when an administrator logs on).

When choosing the icon, a window appears:



The window shows on the left a tree of personalizable aspects. When selecting a node, the aspect can be edited on the right. In our example, we set the "Street (2)" aspect and the "Check ZIP Code" aspect to be invisible. Make sure that the personalization is done within a certain scenario (in this example, the scenario has the name "customer").

After choosing the **Save and Close** button, the address page now looks as follows:

The screenshot shows the 'Address Edit' form. At the top, there is a 'Save' button. Below it, the form is titled 'Address' and contains the following fields:

- First Name
- Last Name
- Street
- Street (2)
- Zip Code
- Town
- Country

At the bottom of the form, there is a 'Check ZIP Code' button.

You see that two lines have disappeared. This definition and the corresponding screen layout are now automatically valid for all users that work in the personalization scenario "customer". Maybe other users work in a different scenario and thus see a different screen layout.

The framework basics will be explained below, and the implementation of the above address example will be shown in detail.

Dynamic Controls

The basis of all personalization is the possibility to define a control's rendering and behavior dynamically via the adapter. Each `BUTTON` control, for example, provides for a `visibleprop` property by which the adapter can specify whether the button is shown or not.

The control of visibility was much extended for personalization purposes. It is possible to control visibility for all major controls including:

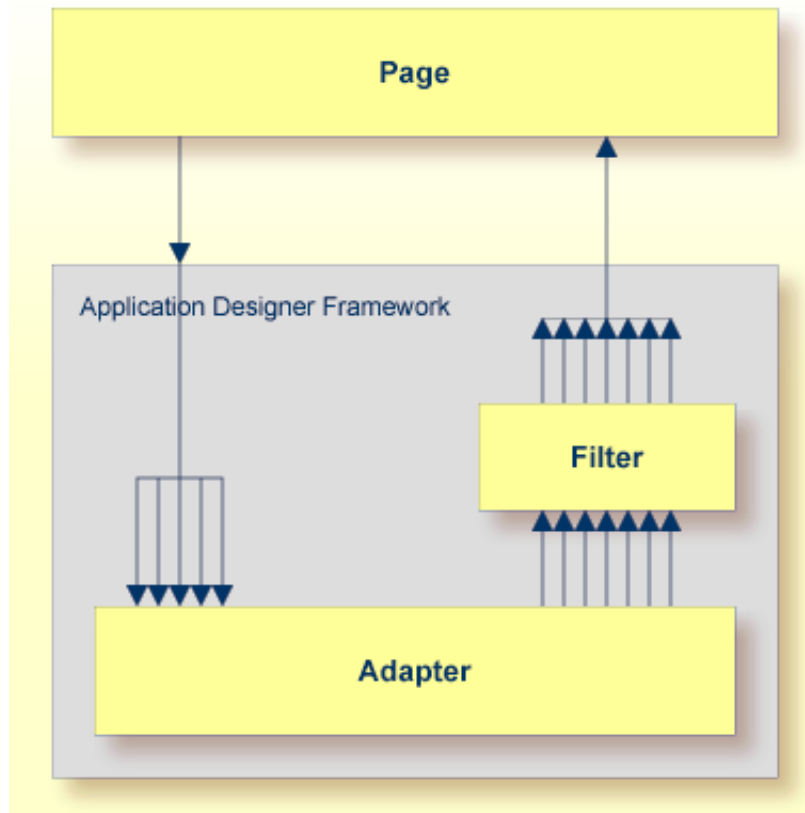
- TR and ITR rows.
- ROWAREA, ROWTABLE0, and other container controls.
- BUTTON, ICON, MENUBUTTON, FIELD, and other input controls

By defining the `visibleprop` properties in the controls, you can - without using the next framework parts - already control the page layout dynamically in a detailed way. It is up to you to specify where and how to use the `visibleprop` definitions. Sometimes it is useful to use the definitions on single elements (e.g. `BUTTON` controls). Sometimes it is useful to define the visibility on row level (ITR, TR control). If inside your screen there is the combination of "label and field in one row", then it makes sense to define visibility on row level.

Using Filters

Having defined the `visibleprop` definitions inside the page, the question now is how to provide for the corresponding property values at runtime.

The concept of filters offers a flexible and efficient way to provide for the values in front of the adapter logic - without even letting the adapter logic know about.



After the Application Designer framework has collected the property values from the adapter, it passes them to a filter that may be assigned to an adapter. The filter can now manipulate the values. This means it can

- change the value of properties,
- add properties that are not provided for by the adapter.

What does this mean for personalization? It means that the adapter does not need to implement all the `visibleprop` properties on its own but can "outsource" this task to a generic filter.

Personalization Filter

The personalization filter is a concrete filter implementation that is responsible for providing personalization data values. On the one hand, the personalization filter is kept well apart from the Application Designer framework; i.e. you will not find any specific personalization aspects inside the base framework of Application Designer (e.g. there are no personalization methods inside the `Adapter` class). On the other hand, it is tightly integrated into the Application Designer framework - being a concrete implementation of open interfaces that are offered.

The main aspects of the personalization filter are:

- Inside the page layout definition, a developer decides at design time which properties are controlled by the filter. These properties are called "managed properties" in the following text.

- At runtime, the filter provides for the property values depending on so-called personalization scenarios: values for the managed properties are kept with reference to a scenario in a persistent storage. This means that the property values are either stored in an SQL database or in the file system.
- At runtime (e.g. when a user logs on to an application system), the application defines the active personalization scenario(s) in the session context. Depending on the scenario(s), the right filter information is read from persistent storage and used by the filter.

Personalization Scenario Sequence

It is possible to define a sequence of personalization scenarios at runtime. Maybe a user is assigned after having logged on to the scenarios "Department 12", "Business Unit1". In this case, the personalization filter first tries to find filter information for "Department 12"; if filter information is not defined, then it tries to find filter information for "Business Unit1".

Consequently, you can set up a flexible way to try to provide for the most detailed personalization definition for specific screens - and referencing to a general personalization definition (or none at all) for other screens.

Maintaining Personalization Data

Personalization data is maintained through a specific page. The maintenance page may be accessed directly from a personalized page via a certain control - a certain icon in the title bar. This icon is only visible when a certain context parameter is set. It should not be visible during normal user sessions, but should only be available for administrative sessions.

Persisting Personalization Data

The personalization data is stored in the local file system, inside the directory */pers* of an Application Designer project.

Caution:

Filter information is runtime information, not design time information. This means: filter information is not part, for example, of a WAR file that you deliver, it is maintained within a delivered system by your customer.