When creating applications in Design Studio, having a strong foundation is important. In this excerpt, you’ll walk through the building blocks of Design Studio: components and their properties.
7.3 Visual Component Properties

In this section, we’ll discuss the properties of all the visual components in Design Studio. Visual components are divided into three categories: analytic, basic, and container components. We’ll discuss the properties for each category, but we’ll begin by discussing the common properties.

7.3.1 Common Properties

The common properties are shared by all visual components. These involve the layout of the components—specifically, their sizes and margins. You can alter the layout and the relative positioning of a component by editing the numbers in the Layout section of the properties. You can also use the mouse to drag the component or its borders to the desired size or position.

Table 7.3 provides a list of common properties in Design Studio.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Sets the unique name of the component. If you don’t enter a name, the system will take a default name, for example, TEXT_32. For maintainability, it’s important to use a naming convention.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Displays the component type (CHECKBOX, LISTBOX, DROPBOX, etc.). You can’t alter this property.</td>
</tr>
<tr>
<td>VISIBLE</td>
<td>Specifies whether a component is visible (True) or not visible (False).</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Specifies whether a component is enabled (True) or disabled (False). Disabled means that the user can see the component, but can’t interact with it.</td>
</tr>
<tr>
<td>CSS CLASS</td>
<td>Sets the CSS class from which the layout properties should be applied to the component. These CSS classes are defined in the CSS file attached to the APPLICATION component (see Section 7.1.1).</td>
</tr>
<tr>
<td>TOP MARGIN</td>
<td>Sets the distance between the top of the component and the top of the parent component. Either enter a number in pixels or choose AUTO.</td>
</tr>
<tr>
<td>LEFT MARGIN</td>
<td>Sets the distance between the left side of the component and the left side of the parent component. Either enter a number in pixels or choose AUTO.</td>
</tr>
</tbody>
</table>

Table 7.3 Common Properties
By setting the layout properties to **Auto**, you’re allowing this property to shrink and grow along with the screen size. When a fixed number is entered for a property, then the value for that property remains constant.

For example, consider a component with the following layout parameters:

- **Top Margin**: 10
- **Left Margin**: 10
- **Bottom Margin**: 10
- **Right Margin**: 10
- **Width**: Auto
- **Height**: Auto

The result of these settings is that the margins will remain equal in relation to the parent component, but the component itself will resize according to the parent component. The example is typical for when you want a background that covers nearly the whole screen with a little margin on the sides.

As another example, consider these parameters:

- **Top Margin**: 10
- **Left Margin**: 10
- **Bottom Margin**: Auto

The result of these settings is that the component size is fixed to 100 × 100 pixels with a margin of 10 pixels to the left and top. A bigger screen size will mean larger right and bottom margins. Typically, these settings are used when you want a logo to appear on the top-left area of the screen.

You can create more advanced settings by combining the settings of container panels. If you place the second example within the first, a **Left Margin** of 10 is relative to the left side of the first component.

### 7.3.2 Analytic Component Properties

Analytic components use data to show numbers in either table or graphical form. Additionally, components in this category enable users to work with the data and alter the way the data is filtered and shown. These components are tied to data by assigning a data source to a component and then using the component to visualize the data output that is defined in the **Initial view** of the data source.

We’ll describe the different types of analytic components next.

#### Chart Component

**Chart** components can be added to an application to visualize data graphically. They can identify trends or outliers in data, and they also help users to focus on those data points. The **Chart** component appears as soon as you assign a data source to it. The default **Chart Type** is a column chart.

A **Chart** component is meant to communicate data in a clear, concise way. The example in Figure 7.12 shows the total population per region from 1990 to 1999. When you hover over a year, you see a tooltip with the exact data value.
Table 7.4 lists the properties of the Chart component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
<td>This assigns data to the component for visualization. You can choose from the defined Data Source Alias components that are added to the application.</td>
</tr>
<tr>
<td>Data Selection</td>
<td>This is a selection string in JavaScript Object Notation (JSON) format. A dialog box allows you to create a subset of the data in the data source to visualize in the Chart component.</td>
</tr>
<tr>
<td>Chart Type</td>
<td>A number of chart types are available, as outlined in more detail later in this section. In Figure 7.13, all the chart types are listed with a screenshot for reference.</td>
</tr>
<tr>
<td>Conditional Formatting</td>
<td>This is a selection string in JSON format. A dialog box allows you to apply conditional formatting rules to measures or dimension members in a chart.</td>
</tr>
<tr>
<td>Swap Axes</td>
<td>This specifies whether to swap the horizontal and vertical axes of the data source before visualization.</td>
</tr>
<tr>
<td>Show Totals</td>
<td>When True, the data source output totals will be visualized.</td>
</tr>
</tbody>
</table>

The types of charts are as follows (see Figure 7.13):

- **Line**
  A line graph shows a trend by showing a line with the labels in the X-axis and the values in the Y-axis. In addition, there are line chart variations:
  - Horizontal line: A horizontal line chart shows the line going in a vertical direction with the labels horizontal.
  - Dual line: Two lines chart where each line has its own axis (one on the left and the other one on the right).

- **Bar**
  A bar chart shows a bar for each value in the data source. There are several bar chart variations:
  - Stacked bar: In a stacked bar chart, you can also show how different values add up to the total of the bar. The length of the stacked bar depends on the sum of the values.
  - 100% stacked bar: This shows bars in which each value is a part of the bar. This type of chart will always show the total bar length at 100%.
  - Bar combination: This is a combination of a bar chart and a line chart.
  - Bar dual axis: This is a two bar series where each series has its own axis.

- **Column**
  A column chart shows each value in a column. There are several column chart variations:
» **Stacked column:** In a stacked column chart, you can also show how different values add up to the total of the column. The total length depends on the sum of the values.

» **100% stacked column:** This shows columns in which each value is part of the column.

» **Column combination:** This is a combination of a column chart and a line chart.

» **Column combination dual axis:** This is a combination of a column chart and a line chart where both have their own axis.

» **Area**
An area chart is like a line chart with the area under the line colored in. In addition, there is a variation of an area chart:

» **Horizontal area:** This is an area chart where the line is vertical and the labels are horizontal.

» **Crosstab**
A crosstab is a table in which numbers are presented along rows and columns.

» **Bubble**
In a bubble chart, you can map three key figures: one on the X-axis, one on the Y-axis, and one that affects the size of the bubble.

» **Waterfall**
A waterfall chart is a bar chart where bars are shown in a cumulative way. The total value of bar 1 is the starting point of bar 2. There are two variations of a waterfall chart:

» **Stacked waterfall:** This waterfall chart has the added ability to add up several values in each bar.

» **Horizontal waterfall:** This waterfall chart shows the bars going from left to right and the labels are horizontal.

» **Pie**
A pie chart shows the relative size of entities compared to the whole. In addition, there is a variation of a pie chart:

» **Multiple pie:** A multiple pie chart shows a different pie graph for each key figure and shows the relative sizes for each value on the graphs.

» **Radar**
A radar chart shows the relative size for each value. It’s like a line graph with a round axis. In addition, there is a variation of a radar chart:

» **Multiple radar:** A multiple radar chart shows a radar graph for each key figure.

» **Scatter**
A scatter chart shows combinations of key figures, where one key figure is plotted along the X-axis, and the other along the Y-axis.
Additionally, you can pick one of the special chart types when you choose ADDITIONAL CHART TYPES. With this option, you have, by default, the option to choose from a trellis, heat map, tree map, and, if you installed the real-time package, the time-based line.

In graphs, you can set up conditional formatting, which enables you to highlight a particular item when a condition is met. For example, you can highlight all regions with red that missed their targets by at least 5%.

In the following steps, we’ll walk through how to apply this format based on our example:

1. Select the GRAPH component for which you want to add conditional formatting.
2. In the GRAPH component’s properties, click the CONDITIONAL FORMATTING button.
3. If you want to create a new rule, on the CONDITIONAL FORMATTING screen, click the NEW RULE button. The NEW RULE dialog box appears (Figure 7.14). If you click the EDIT RULE button, the EDIT RULE dialog box appears (Figure 7.15). Both the NEW RULE and the EDIT RULE screens are identical. Therefore, whether you’re creating a new rule or editing an existing rule, enter or edit the name of your rule in the NAME field.
4. Under RULE DESCRIPTION, select either a MEASURE or a DIMENSION MEMBER from the dropdown list. When you select MEASURE, you’ll be able to highlight based on the evaluation of a value. If you select DIMENSION MEMBER, you can highlight cells that belong to that member. If you choose MEASURE, the screen will look like Figure 7.14. If you choose DIMENSION MEMBER, the screen will look like Figure 7.15.
5. For the MEASURE, select the type of condition (EQUAL TO, IS GREATER THAN, IS LESS THAN). Select IS GREATER THAN, as shown in Figure 7.14.
6. Input the value threshold that the measure value is compared to, for example, “100,000.” With this input, you’ve instructed the application to highlight every cell in the measure that has a value greater than 100,000.
7. In the PREVIEW area, click the FORMAT button to assign a color to your rule.
8. Click OK to finish customizing your rule.

Your new formatting rule is now listed in the CONDITIONAL FORMATTING dialog box. As previously discussed, if you want to edit your rule, on the CONDITIONAL FORMATTING screen highlight, the rule you want to edit, and click EDIT RULE. Use the up and down arrows to change the order of the rules on the CONDITIONAL FORMATTING screen.

In the ADDITIONAL PROPERTIES tab (Figure 7.16) of a chart, you can alter the look of the CHART component. In the CHART AREA, you can choose which labels to make visible, whether you want a tooltip (a message that appears when the users hovers the mouse over the graph), and whether a chart animation should be
Components and Properties

shown at application startup. You can also choose the fill color of your background and how the dimension values are displayed in the chart.

Figure 7.16 Additional Properties of a Chart Component

With the Data Series Format additional property of a chart (Figure 7.17), you can set the colors for all the data series. Keep in mind, however, that the colors are attached to the index number, not to the data series name. Therefore, if you change the order of the data series, the colors will change accordingly.

Figure 7.17 Data Series Properties

The CSS additional properties (Figure 7.18) allow you to format different parts of the chart, specifically the CSS settings. By changing fonts, sizes, and so on, you can control how labels and titles are formatted within your application. Be aware, however, that these settings only apply to the component you’re editing. If you want the settings to apply to all charts, you have to work with custom CSS.

Figure 7.18 Subset of the CSS Additional Properties
Info Chart Component

New to Design Studio 1.6, the Info Chart component (Figure 7.19) is the future replacement for the Chart component. One of the main differences between the Info Chart component and the Chart component is the way you configure the chart. For the Info Chart component, you can use the Chart Configuration property to open the Configure Chart screen for setting up your chart (Figure 7.19).

![Figure 7.19 Info Chart Configuration](image)

On the Configure Chart screen, you can choose the type of graph you want to use under Chart Type. On the left side of the screen, you see the layout of the data source, where you can change the layout for this graph. Here, you can work with all the dimensions that have been added to the rows or columns in the data source.

Follow these steps to configure an Info Chart component:

1. Assign a data source to your Info Chart component.
2. Within the Properties tab, select the current value for the Chart Configuration property, which opens a screen for further configuration.

3. Select the Chart Type above the rendered chart. Under Chart Type are seven collections of charts that you can choose from. The selected graph will change the example graph below the picker.
4. Move the measures and dimensions up and down within their areas to configure the binding of data to the Info Chart component. Changes influence the chart directly.
5. Select OK. The Chart Type and data binding are now applied to the Info Chart component. The newly configured Info Chart is displayed in the Layout Editor.
6. To display conditional formatting, set the Conditional Formatting Visible property to True.
7. To display the totals in your charts, set the Show Totals property to True.

Info Chart Feeding Panel

The Info Chart Feeding Panel (Figure 7.20) allows users to change the layout of the Info Chart at runtime. Users can drag the dimensions that will influence the appearance of the chart. The Info Chart component doesn’t influence the data source, which means changes only will apply to the related graph.

In the Properties tab, you have to define which graph it should influence. Other graphs will remain unaffected.

![Figure 7.20 Chart Feeding Panel with the Info Chart](image)
Flexibility

The Info Chart Feeding Panel allows you to set the data source of an Info Chart component and design an application for a single graph and change the dataset and layout at runtime. By doing this, you can create applications that allow users to explore and choose their own graph types. At startup, you can create a new application with the Data Discovery and Visualization template.

Scorecard Component

The Scorecard component (Figure 7.21) is used to create a table with numerous graph options. You can create tables with graphs in each row visualizing that particular dataset. This is similar to the trellis function in SAP Lumira.

Warning!

Given the complexity of this component, make sure you have a separate data source for this component and that the Initial view is set properly. If you have to change either of these options later on, chances are you’ll have to redo the configuration entirely.

3. Select the Scorecard component, and go to the Properties tab.
4. Set the row and header heights.
5. Set the Selection Type (Single or Multi-Toggle).
6. Set the Navigation Mode (Navigator or Scroll).
7. Go to the Additional Properties tab (Figure 7.22).

8. Select the Dimension for Row Scope. This will determine which dimension will group the scorecard into rows.
9. Click the Generate Initial Scorecard button. A first rendition of the Scorecard is shown in Figure 7.23.

When you click the button, the system generates the following columns:

- For every dimension included in the rows: One column with dimension members.
- For the first measure: One dimension displaying the measure sum of the row scope definition.
- For the first dimension, which is outside of the row scope: A column with a trend chart displaying the measure members for that dimension as a line or column chart.
- If the data source has more than one dimension: A column with a comparison chart displaying the delta between the first and second measure.
10. Select the Scorecard, and go back to the Additional Properties tab. The Additional Properties are now expanded to include columns (Figure 7.24). These columns relate to the columns you see in the initial rendering.

11. In the Additional Properties tab, you now can select columns. The selected column can be configured below the selection.

12. Select the Cell Template (text or graph type). The other selections below will change based on the selection made for the Cell Template.

For our example, you’ll configure a text column and a graph column. In the two subsections that follow, we’ll walk through each of these processes.

**Text Column**

Some properties are bound to data and some aren’t. You can tell whether a property is bound to data by the icon on the right side of the property, as shown in Figure 7.25 and Figure 7.26. Bound implies that the value of the cell is dependent on the underlying data source. Unbound implies the cell values are independent values. Figure 7.25 shows an example of a bound property, and Figure 7.26 of an unbound property.

By clicking on the icon on the right side of the property, you can change the property. For unbound properties, you can insert manual entries. For bound properties, you can link to parts of the data source.

To edit a text column in a Scorecard component, follow these steps:

1. Go to the Header Text section, and click on the Bound property.

2. Open the Binding Type dropdown menu to see the available options. Select Dimension Metadata Content.

3. Select the Dimension.

4. Select Value Text or Value Key for the Dimension Display. The first is the unique technical value of the dimension member, and the second is the text description. If you want to replace the value of the property with your own text:
Change the Binding Type to Unbind, Manual Entry.
Type your own text into the input box.

5. Go to Cell Content – Text.
6. Open the Binding Type dropdown menu to see the available options. Select Dimension Member Content.
7. Select the Dimension.
8. Select the way the value is presented in the Member display. Select either Value text or Value key for Dimension Display.
9. Select which part should be cut off if the member text is too long in the Member Presentation Part property. Options include Default, First Part, or Last Part.
10. In the other tabs, set the layout for both the colors and margins.
11. In the Column Properties tab, set the width of the column.

Graph Column
Now you’re going to configure a graph column. First, you’ll set up the graph type. Then, you’ll bind the values and configure the labels. There are many more options, however, this will get you through the first steps. Follow these steps:

1. Select a column that contains a graph. Note the Cell Template; it states the type of chart used.
2. Go to the Chart Size property. Choose Chart with Data Labels if you want labels included.
3. For the Actual Values property, click the Bound icon.
4. In the Bound Type field, select Multiple Cell Count.
5. If you click the icon on the right side of the selected data cells, you’ll see a pop-up where you can define the cell set you want to visualize (Figure 7.27).
6. Repeat the steps for each column. First, select a template, and from there, adjust the individual properties.

Crosstab Component
The Crosstab component (Figure 7.28) is useful for displaying detailed multidimensional data for analytical purposes.

![Figure 7.27 Data Selection for a Column in the Scorecard Component](image)

![Figure 7.28 Crosstab Component](image)
Used together with the Dimension Filter component and the Filter Panel component, which will be described later in this chapter, a Crosstab component is a very flexible way to show and work with data. You can use the Crosstab component to sort and filter data, move or swap dimensions, and select cells that in turn can be scripted to add filters to data sources.

Table 7.5 shows the properties of the Crosstab component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA SOURCE</td>
<td>Assigns data to the component to visualize it. You can choose from the defined DATA SOURCE ALIAS components you added to the application.</td>
</tr>
<tr>
<td>Pixel-Based Scrolling</td>
<td>Enables a smooth scrolling experience. This property is recommended when building an application for mobile devices or applications with a low data volume.</td>
</tr>
<tr>
<td>Row Limit</td>
<td>Sets the maximum number of rows for pixel-based scrolling.</td>
</tr>
<tr>
<td>Column Limit</td>
<td>Sets the maximum number of columns for pixel-based scrolling.</td>
</tr>
<tr>
<td>Selection Type</td>
<td>Sets whether the user can select a row, a column, or single data cells.</td>
</tr>
<tr>
<td>Selectable Area</td>
<td>Sets the area of the Crosstab where selection is allowed.</td>
</tr>
<tr>
<td>Enable Hover Effect</td>
<td>Enables a hover effect when selection is allowed.</td>
</tr>
<tr>
<td>Hierarchy Navigation</td>
<td>Specifies whether the user can collapse or expand hierarchies in the Crosstab.</td>
</tr>
<tr>
<td>Sorting Enabled</td>
<td>Specifies whether the user can sort columns in the Crosstab.</td>
</tr>
<tr>
<td>Column Resizing Enabled</td>
<td>Specifies whether users can resize columns in the Crosstab.</td>
</tr>
<tr>
<td>Horizontal Resizing</td>
<td>Enables or disables resizing of horizontal headers.</td>
</tr>
<tr>
<td>Horizontal Scrolling</td>
<td>Enables users to scroll horizontally for headers.</td>
</tr>
<tr>
<td>Context Menu Enabled</td>
<td>Specifies whether the user can open a context menu in the Crosstab by right-clicking. You're also able to modify the menu items in the context menu.</td>
</tr>
<tr>
<td>CSS Class</td>
<td>Assigns a class when a custom CSS file is assigned to the APPLICATION component’s properties.</td>
</tr>
</tbody>
</table>

Table 7.5 Crosstab Component Properties

As an example of when this component can come in handy, you can use the Crosstab component in combination with a Chart component. Users can click a value on the Chart component, and the Crosstab component will show detailed...
Components and Properties

information about the selected value. The selected value then can be used as a filter for a second component for a more detailed view.

For this example, let's assume that a second data source is being filtered based on the selection made in the Crosstab component. This results in an interactive screen that will pop up with additional information when a cell is being selected. Figure 7.29 shows an application where the user selected the decade 1950–1959 from below the graph, and a Crosstab component appeared to show each year in that decade.

Figure 7.29  Click Through from Graph to Crosstab

The Crosstab component is placed inside a Panel component that isn’t visible at the start of the application. When the user clicks on a line in the chart, a filter is applied to the second data source, and the Panel component is set to Visible.

The code for this is shown in Listing 7.4.

```javascript
var currentMembers = CHART_1.getSelectedMembers("REGION_NAME");
if (currentMembers.length > 0) {
    DS_2.setFilter("REGION_NAME", currentMembers);
     PNL_COUNTRIES.setVisible(true);
} else {
    PNL_COUNTRIES.setVisible(false);
}
```

Additionally, you see a conditional statement in the example script. In this statement, you check how many members are selected in the chart. If any are selected, the panel with the Crosstab is made visible, and the filter is applied. If no selection is made, the Crosstab disappears again.

Spreadsheet Component

The Spreadsheet component displays data in a grid (Figure 7.30). The main advantage the Spreadsheet component has over the Crosstab component is that it has easy-to-use copy and paste functions. These functions are useful in planning scenarios because employees can use data that they’ve gathered in their Excel spreadsheets and directly copy them to the Spreadsheet component.

Figure 7.30  Spreadsheet Component
When using the Spreadsheet component, there are a few limitations:

- You can’t use the universal display hierarchy of SAP BW data sources. Therefore, opening and closing nodes isn’t possible.
- Only one hierarchy per axis is supported.
- The key and text of dimensions are only displayed in one cell/column.
- Attributes aren’t supported.
- In SAPUI5 rendering mode, the Spreadsheet component is only supported in the Compact Form Factor.
- The Spreadsheet component can only be used with the Blue Crystal theme.
- The Spreadsheet component doesn’t allow any interaction in mobile applications.

When using the Spreadsheet component in Design Studio on SAP HANA, there are more limitations:

- Bottom-up hierarchies aren’t supported.
- New lines can’t be inserted.
- Dynamic calculations and calculations based on the menu entry Calculate Totals aren’t supported.

In most circumstances, the Crosstab component may be a better option due to these limitations. A Spreadsheet component is only a good choice when you have a scenario where copying and pasting numbers for planning is required.

**Filter Line Component**

The Filter Line component (Figure 7.31) offers the user a list of filters that are applied to the assigned data source.

When a custom CSS file is assigned to the Application component’s properties, you can assign a class here.

You can specify whether the user is able to filter measures.

Table 7.6 lists properties for the Filter Line component.
Dimension Filter Component

The **Dimension Filter** component (Figure 7.34) is useful for adding a filter for one dimension. This filter can be applied to multiple data sources. When clicked at runtime, the component opens a popup in which the user is able to select a value or a range of values. In the **Selection** tab, you can choose filter values by selecting the members in the table, and you can limit the number of possible choices by entering a search string in the textbox. For example, if you want to choose from all the products starting with N, enter “N*” in the text box.

![Figure 7.34 Dimension Filter Component](image)

In the **Range** tab, you can build a range filter with a lowest and highest value. First, select the **Starts At** row, and then select the value. Then select the **Ends At** row, and select the highest value. Click the **Add to List** button on the bottom of the popup screen. Now you can make a new range filter and add it to the list. When you’re finished, click the **Apply** button at the top-right area of the screen. When you click **Back**, the last filter you entered will be ignored.

When you want the user to be able to filter on more dimensions, you have to add a **Dimension Filter** component for each dimension to which you want to allow filtering for. (Alternatively, you could use the **Filter Panel** component, which is discussed next.)

Table 7.7 shows the properties of the **Dimension Filter** component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Source</strong></td>
<td>This assigns data to the <strong>Dimension Filter</strong> component. This data source delivers the items for which you can set a <strong>Dimension Filter</strong> on the target data sources defined in the next property.</td>
</tr>
<tr>
<td><strong>Target Data Sources</strong></td>
<td>With this property, the <strong>Dimension Filter</strong> component can be applied to other data sources. Data sources must have the same dimension and must be defined in the application. If you have an application that shows sales, purchases, the general ledger, and transport for regional offices, you have several data sources for each dataset. One <strong>Dimension Filter</strong> component on a regional office dimension would set the correct filter on all these data sources.</td>
</tr>
<tr>
<td><strong>Dimension</strong></td>
<td>This is the dimension to be filtered.</td>
</tr>
<tr>
<td><strong>Dimension Name</strong></td>
<td>This is the dimension name.</td>
</tr>
<tr>
<td><strong>Display Mode</strong></td>
<td>This property sets the way filters are displayed:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Filter List</strong>: Filter values are displayed as comma-separated values.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Filter Count</strong>: The number of applied items is displayed.</td>
</tr>
<tr>
<td><strong>Member Display</strong></td>
<td>This specifies the way the dimension members are displayed.</td>
</tr>
<tr>
<td><strong>Popup Width/Height/Position</strong></td>
<td>This is the layout of the <strong>Popup</strong> component that is used to define the <strong>Dimension Filter</strong> component.</td>
</tr>
<tr>
<td><strong>Popup Is Modal</strong></td>
<td>If set to True, the user can’t interact with any other part of the application.</td>
</tr>
<tr>
<td><strong>Remove Redundant Selections</strong></td>
<td>This is for hierarchical dimensions. If set to True, the system will remove redundant selections, for example, a node and a subnode within.</td>
</tr>
<tr>
<td><strong>Auto Apply</strong></td>
<td>This specifies if the changes made by the user are applied automatically.</td>
</tr>
<tr>
<td><strong>On Apply</strong></td>
<td>This is the script that is executed when a filter is applied. A Script Editor opens for this property.</td>
</tr>
</tbody>
</table>

Table 7.7  **Dimension Filter Properties**
When you use multiple Dimension Filter components, you can limit the usage by hiding the components that can’t be used. For example, if the user already filtered on products, you could stop him from also filtering on customers by hiding the Dimension Filter component that handles the customer filter by using the following code:

```javascript
DIMENSIONFILTER_CUSTOMER.setVisible(false);
```

The best part about the Dimension Filter component and the Filter Panel component is that filters can be applied to multiple data sources. This can be very useful if you need to keep multiple data sources in sync. The user performs only one filter action, and all data sources have the same filter.

### Filter Panel Component

The Filter Panel component (Figure 7.35) allows you to apply filters on several dimensions to target data sources without having to resort to scripting. The Dimension Filter component, discussed previously, can also be used to apply filters to target data sources, but the Filter Panel component can put filters on more than one dimension.

![Figure 7.35 Filter Panel Component](image)

The Filter Panel component shows all the dimensions of the data source to which it’s assigned. The user can open an input box by clicking the name of the dimension. By clicking the - sign on the right, the user can open a selection screen to pick values.

Table 7.8 shows the properties of the Filter Panel component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
<td>This assigns data to the component for filtering. You can choose from the defined Data Source Alias components added to the application.</td>
</tr>
<tr>
<td>Target Data Sources</td>
<td>With this property, the Filter Panel component can be applied to other data sources. Data sources must have the same dimension and must be defined in the application.</td>
</tr>
<tr>
<td>Mode</td>
<td>This enables you to set the way the component works:</td>
</tr>
<tr>
<td></td>
<td>Filtering: Enables the user to filter data.</td>
</tr>
<tr>
<td></td>
<td>Navigation: Allows the user to add or remove dimensions from rows or columns.</td>
</tr>
<tr>
<td></td>
<td>Filtering and Navigation: Enables both functions at once.</td>
</tr>
<tr>
<td>Dimensions</td>
<td>This selects and orders the dimensions that will be available for the user.</td>
</tr>
<tr>
<td>CSS Class</td>
<td>When a custom CSS file is assigned to the APPLICATION component, you can assign a class here.</td>
</tr>
<tr>
<td>Dimension Name</td>
<td>This shows the dimension name.</td>
</tr>
<tr>
<td>Display Mode</td>
<td>This shows the filters or the number of filters applied.</td>
</tr>
<tr>
<td>Member Display</td>
<td>This sets member display mode, for example, key + text.</td>
</tr>
<tr>
<td>Title</td>
<td>This sets a title for the Filter Panel component.</td>
</tr>
<tr>
<td>Remove Redundant Selections</td>
<td>This is for hierarchical dimensions. If set to True, the system will remove redundant selections, for example, a node and a subnode within.</td>
</tr>
<tr>
<td>Drag &amp; Drop</td>
<td>This specifies whether the user can interact using drag and drop.</td>
</tr>
<tr>
<td>Direct Input for Filter</td>
<td>This specifies whether the user can use direct input for dimension or measure keys.</td>
</tr>
<tr>
<td>Auto Apply</td>
<td>This specifies whether all the changes made in the component are applied automatically.</td>
</tr>
<tr>
<td>Maximum Number of Members</td>
<td>This specifies the maximum amount of members in the value help. If the actual number exceeds the maximum, no members are displayed. Instead, the user has to limit the number of members by searching for more specific text or a key.</td>
</tr>
</tbody>
</table>

Table 7.8 Filter Panel Component Properties
One of the things you can do using the Filter Panel component is control which dimensions you display. Although you could put all the dimensions in one Filter Panel, it’s easier for the user of the application when you put dimensions that belong together in one Filter Panel. For example, in an application that shows sales data, you can put all the customer dimensions in one Filter Panel, the product dimensions in the second, and the time dimensions in the third panel. You could also put a couple of Filter Panel components in one Panel Container component and, for example, allow the user to toggle the visibility of these Panel components. That way, you can create a dimension-like menu structure in which the user can navigate.

The Filter Panel component already has interactivity built into it because it interacts with the data source. You can, however, add something to the layout to highlight that this component has been used to add a filter. In the On Apply handler, set the following code:

FILTERPANEL_1.setCSSClass("Active");

This will assign a CSS class with, for example, a different font color to signal that this filter has been used.

Geo Map

The Geo Map component allows you to display layers of information on top of a map and allows users to drill down through the different layers (Figure 7.36). You can create three types of layers: shapes, points, and bubble charts.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Apply</td>
<td>This script is executed when a Filter Panel component is applied. A Script Editor opens for this property.</td>
</tr>
<tr>
<td>On Cancel</td>
<td>This script is executed when the Cancel button is clicked. A Script Editor opens for this property.</td>
</tr>
</tbody>
</table>

Table 7.8  Filter Panel Component Properties (Cont.)

The Geo Map component properties are described in Table 7.9.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basemap URL</td>
<td>This is the URL to retrieve the basemap. The basemap is the map rendered underneath the graphical representation of the data.</td>
</tr>
<tr>
<td>Basemap Copyright</td>
<td>This is the copyright text for the basemap tiles.</td>
</tr>
<tr>
<td>Map Legend Visible</td>
<td>This specifies whether the map legend is visible.</td>
</tr>
<tr>
<td>On Select</td>
<td>This is triggered when the selected item on the map is changed.</td>
</tr>
</tbody>
</table>

Table 7.9  Geo Map Component Standard Properties
The layers on top of the map are created in the Additional Properties tab (Figure 7.37). In this tab, you can create multiple layers that will be placed on top of the map.

For each layer, you can set a number of properties that will influence the way that the layer will appear on the map (see Table 7.10).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>This is the unique ID for a layer. This ID is also used for the <code>setLayerVisible()</code> method that will allow users to show/hide layers by interacting with the application.</td>
</tr>
<tr>
<td>Show Layer</td>
<td>This is the initial setting to show or hide the layer on the map.</td>
</tr>
<tr>
<td>Type</td>
<td>This is the type of layer. There are three settings: SHAPES, POINTS, CHARTS.</td>
</tr>
</tbody>
</table>

Table 7.10 Additional Properties for Geo Maps

7.3.3 Basic Component Properties

In this section, we’re going to look at the basic components, which are BUTTON components, TEXT components, and IMAGE components. They don’t visualize the content of a data source or allow child components. There are some options to bind data sources to properties to set the properties automatically instead of via script.

Data Binding

Basic components’ properties now have the ability to use a data binding function (Figure 7.38) to set the property value. This feature allows you to create applications with a lot less scripting because many properties are automatically updated if the underlying data source changes. Previously, you had to transfer those values using script.
There are three types of binding:

- **Data Cell Binding**
  This allows you to retrieve the numerical value of a single cell from your data source.

- **Dimension Members Binding**
  This allows you to retrieve a list of dimension members.

- **Filter Binding**
  This allows you to use the selected data and automatically create a filter on another data source.

The second and third options are typically used for components such as the List Box component, Dropdown component, and the Checkbox Group component to get a list of options.

You can use Data Cell Binding for many properties. Data Cell Binding looks at a single cell value in the data source. In the binding, you first select the data source, then select the cell within the data source, and finally you create a formatter function that will take the cell as an input and enable you to change it with script. For item properties, you can use the Dimension Members Binding where you can bind to a dimension so the property will get all the members of that dimension. Finally, Filter Binding is an automated task that you can create based on the selected value in the component. You can automatically set the Filter Binding to apply a filter to a data source.

Now let’s look at a step-by-step example for a Dropdown component. Here, you want to have all the regions in the items and apply a filter when a region is selected. Follow these steps:

1. First, drag a radio button group onto the canvas (Figure 7.39).
2. Select the **Items** property, and click on the arrows icon on top of the Properties pane (Figure 7.40).
3. Select both the **Dimension Members Binding** and the **Filter Binding** options (Figure 7.41).
4. In the Properties pane, you can now set the data binding options (Figure 7.42).
5. Provide values for the **Data Source**, **Dimension**, and **Member Display** fields for the input binding (Figure 7.43).

6. Now go to the **Output Filter Binding** properties, and set the **Data Source** that should be filtered. When clicking on the **Data Source**, the **Select Target Data Sources** screen appears, where you can select multiple data sources (Figure 7.44).

7. Finally, set the **Dimension** on which the filter must be applied. (Figure 7.45).

With this feature, you can create an action in response to the selection of one or more of the items listed in the component. The action will change the other components based on your selection. This way, you don’t have to write additional scripts.

The ability to add **Filter Binding** is available in any component that has an item property. With **Data Cell Binding**, there is an option to format the value before it’s applied to the property using the **Formatter Function** (Figure 7.46). Besides formatting, this function can also perform any action on the value you like. For example, you might use the returned number value to look up a text.

Buttons allow the user to interact with the application (Figure 7.47). To build this interactivity, you add a script to the **Button component’s On Click** property.
Table 7.11 shows the properties of the Button component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>This specifies the text displayed on the Button component. It’s possible to display text, icons, or a combination of both.</td>
</tr>
<tr>
<td>Icon</td>
<td>This specifies the icon to be displayed on the Button component. If the icon image is in the application directory, providing the file name is sufficient. When the file is located somewhere else, click the ... button to the right of the text box of the property to navigate to the image. If the image is located on the Internet or intranet, you can use a URL, for example, <a href="http://www.imageserver.com/myimage.jpg">http://www.imageserver.com/myimage.jpg</a>.</td>
</tr>
<tr>
<td>Tooltip</td>
<td>In this property, enter a message that will show up when a user hovers the mouse over the button.</td>
</tr>
<tr>
<td>CSS Class</td>
<td>When a custom CSS file is assigned to the Application component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>On Click</td>
<td>This pens the Script Editor to add user interaction.</td>
</tr>
</tbody>
</table>

Table 7.11  Button Component Properties

Having the appearance of the Button component itself change when the user clicks the button helps the user to understand what is happening in the application. For example, if the user switches to another screen and comes back five minutes later, it’s immediately apparent where he is in the application.

For a simple example, if you have a Button component that, when clicked, excludes internal sales, it would be helpful if the Button component showed that state. When you look at the examples in Figure 7.48, it’s clear that Intercompany sales and Delivered not invoiced sales are included, but Internal sales and Sold non delivered aren’t.

To make the switch between the enabled state and the disabled state possible, the script checks the Button component’s current state and switches to the other state. The script for our internal sales example is shown in Listing 7.5.

```java
if (BUTTON_INTERNAL.getText() == "Include Internal Sales") {
    BUTTON_INTERNAL.setText("Internal Sales included");
    BUTTON_INTERNAL.setCSSClass("Included");
} else {
    BUTTON_INTERNAL.setText("Include Internal Sales");
    BUTTON_INTERNAL.setCSSClass("Included");
}
```

Listing 7.5  Script to Switch between States

To further emphasize the state, we’ve added a script line to set the CSS class of the text object depending on its current state, so the user can see the state based on the appearance of the button.

Chart Type Picker Component

The Chart Type Picker component allows a user to choose a different graph type to visualize the data. An application designer is able to add additional visualization types to the existing palette (Figure 7.49).

In the Chart Type Picker component, the properties shown in Table 7.12 are included.
Checkbox Component

CHECKBOX components help the user interact with the application (Figure 7.50). CHECKBOX components can be used as on/off buttons to support other components. If you have multiple options, it’s easier to use the CHECKBOX GROUP component (Figure 7.50). The CHECKBOX component is useful when you don’t want to present all checkbox options as a single group.

Table 7.13 shows the properties for the CHECKBOX component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>Specifies the text displayed on the CHECKBOX component.</td>
</tr>
<tr>
<td>SELECTED</td>
<td>Specifies whether the CHECKBOX component is initially selected.</td>
</tr>
<tr>
<td>TOOLTIP</td>
<td>Specifies the message that will be shown when the user hovers the mouse over the checkbox.</td>
</tr>
<tr>
<td>ON CLICK</td>
<td>Opens the Script Editor to add user interaction.</td>
</tr>
</tbody>
</table>

Table 7.13 CHECKBOX Component Properties

To understand how the CHECKBOX component can improve an application, look at Figure 7.51. In this example, you can see the CHECKBOX GROUP component on the bottom of the screen. You use the CHECKBOX component to manipulate the graph so that it only shows one of the three key figures based on the choice of CHECKBOX component. By using an On Click event for each CHECKBOX component, you can select and deselect key figures and see the graph change accordingly. The result is that the graph on top will only show those lines for which the checkbox on the bottom is selected.

Figure 7.50 Example Checkbox Component

Table 7.12 Properties of the Chart Type Picker

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHART REFERENCE</td>
<td>This is the chart that will change based on the selection.</td>
</tr>
<tr>
<td>ADDITIONAL TYPES</td>
<td>You can add or remove additional visualization types for the CHECKBOX PICKER component.</td>
</tr>
<tr>
<td>ON SELECT</td>
<td>This script is triggered when the selection has been changed. In the script, it’s possible to perform additional changes to the chart based on the selected CHART TYPE that you can find using the getSelectedValue() method.</td>
</tr>
</tbody>
</table>

Table 7.12 Properties of the Chart Type Picker

You might also imagine more complicated scenarios where some choices make other options unfeasible. An example is when a user has multiple CHECKBOX components to filter the data source. If you want to avoid a situation where the user applies a filter that results in zero records, you can disable all the components that will lead to this result. Using the enabled and selected property in the script, you can manage the CHECKBOX components to reflect those scenarios.

In the On Click property of CHECKBOX_1, the script would look like this:

CHECKBOX_1.setChecked(true);
CHECKBOX_2.setChecked(false);
CHECKBOX_3.setChecked(false);

Checkbox Group Component

The CHECKBOX GROUP component is an extension of the CHECKBOX component because it’s able to show a list of checkboxes in one component (Figure 7.52).
Just as with other components, the Checkbox Group component can be populated using script or data binding. In script, you can get an array of selected values (all the checkboxes that have been selected).

The properties of the Checkbox Group component are listed in Table 7.14.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEMS</td>
<td>List of items that can be completed by manual input, data binding, or script.</td>
</tr>
<tr>
<td>ON SELECT</td>
<td>The script that is run when the selection of items has been changed.</td>
</tr>
</tbody>
</table>

Table 7.14 Checkbox Group Properties

The Checkbox Group component can be used to apply filters to data sources. Using the group, it's very easy to select multiple members of a dimension and pass that list to the filter.

To use the selected items to create a filter, the following script will suffice in the On Select event of the Checkbox Group component:

```javascript
var currentSelection = CHECKBOXGROUP_1.getSelectedValues();
DS_PRODUCTION_OUTPUT.setFilter("0CALMONTH", currentSelection);
```

The output in the getSelectedValue() method is an array of key values that are passed as filter values to the data source. For readability, it's split in two lines where the list is assigned to the variable currentSelection.

Instead of using the script, you also can use data binding to create an outbound bind to the ITEMS property. This will yield the same result as the preceding script.

**Date Field Component**

The Date Field component enables the user to select a date (Figure 7.53). The entered date can be used in other parts of the application. To enable this interactivity, a script has to be added to the On Select handler.

The main use for the Date Field component is to set a date, which is then used to filter a data source dimension that holds calendar day values. The statement to do this is as follows:

```javascript
Var SelectedDate = DATEFIELD_SALESDATE.getDate()
DS_SALES.setFilter("0CALDAY", SelectedDate);
```

**Dropdown Box Component**

The Dropdown Box component enables the user to select items from a list (Figure 7.54). A common use for this component is setting a filter.
Components and Properties

7

272

Figure 7.54  Dropdown Box Component

Table 7.16 shows the properties of the Dropdown Box component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS CLASS</td>
<td>When a custom CSS file is assigned to the APPLICATION component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>ITEMS</td>
<td>With this property, the items available in the Dropdown Box component can be edited. A key has to be entered for each value. Providing a text label and setting a default item is optional. Items can be added manually at design time, via script, or via data binding.</td>
</tr>
<tr>
<td>ON SELECT</td>
<td>This opens the Script Editor to add user interaction.</td>
</tr>
</tbody>
</table>

Table 7.16  Dropdown Box Component Properties

A common way to set the items you can select in a Dropdown Box component is to populate the items of the component at runtime. To do this, you can add a script to the APPLICATION component at the On Startup handler:

```
Var calendarMonths = DS_SALES.getMemberList("0CALMONTH", MemberPresentation.EXTERNAL_KEY, MemberDisplay.KEY_TEXT, 20);
DROPDOWN_1.setItems(calendarMonths);
```

Script Readability

You may already have noticed that variables are used to store the values for use in the script. Although the script will work fine without using a variable, we still advise you to do this because it enhances the readability of the script, something you’ll appreciate when you have to revisit the script later. Chapter 11 provides more details on design principles.

This script instructs the application to fill the items of **DROPDOWN_1** with the **0CALMONTH** dimension of the data source **DS_1**.

Using the actual values in the data source ensure that the DROPDOWN BOX component only holds the values that are actually available in the data source. This avoids situations where a user applies a filter, and the result is an empty dataset. In this example, the current year might not have ended yet so not all months are available. In this situation, there will be fewer than 12 months to choose from.

Formatted Text Field Component

A FORMATTED TEXT FIELD component allows you to format the text (Figure 7.55). In the ADDITIONAL PROPERTIES panel, you’ll find a simplified text editor with some extra options to add formatting. Additionally, you can assign HTML tags. In the custom CSS file, you can create classes that allow for some extra formatting.

![Figure 7.55 Formatted Text Additional Properties](image)

The FORMATTED TEXT FIELD component is configured in the ADDITIONAL PROPERTIES, where you’ll find the CSS CLASS property to assign a class if you want to apply extra formatting.

Fragment Gallery Component

With the FRAGMENT GALLERY component, a user can see fragment bookmarks (Figure 7.56). The user can then drag bookmarks into a SPLIT CELL CONTAINER component to compose his dashboard.
Components and Properties

Figure 7.56  Fragment Gallery with the Split Cell Component

Table 7.17 lists the properties of the *Fragment Gallery* component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM DIMENSION</td>
<td>Specifies the value in pixels from which the width or height (depending on</td>
</tr>
<tr>
<td>DISPLAY MODE</td>
<td>Specifies the display mode:</td>
</tr>
<tr>
<td></td>
<td>▶ TEXT</td>
</tr>
<tr>
<td></td>
<td>▶ IMAGE</td>
</tr>
<tr>
<td></td>
<td>▶ IMAGE/TEXT</td>
</tr>
<tr>
<td>ORIENTATION</td>
<td>Shows whether to display the fragments vertically or horizontally.</td>
</tr>
</tbody>
</table>

Table 7.17  Fragment Gallery Component Properties

**Icon Component**

The *Icon* component (Figure 7.57) is used to show icons based on a font file. The standard choices delivered by SAP already give you many options. You can add your own by adding a font file.

Table 7.18  Icon Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOM FONT</td>
<td>Name of the font file where you can add your own icons.</td>
</tr>
<tr>
<td>COLOR</td>
<td>Color of the image.</td>
</tr>
<tr>
<td>BACKGROUND COLOR</td>
<td>Color of the background.</td>
</tr>
<tr>
<td>SIZE FACTOR</td>
<td>Relative size of icon in relation to the background.</td>
</tr>
<tr>
<td>ICON URI</td>
<td>Selection of icon in the font.</td>
</tr>
<tr>
<td>TOOLTIP</td>
<td>Text shown when mouse cursor hovers over icon.</td>
</tr>
</tbody>
</table>

**Image Component**

With the *Image* component, you can enhance the layout of the application (Figure 7.58). The *Image* component also has a number of properties for interactivity purposes. One of the most useful properties is the ability to change the *Image* component at runtime, which means you can change the look of the application based on the data values or as the result of specific user interactivity actions. *Image* components can also respond to hovering and clicking.
Table 7.20 shows the properties for the IMAGE component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS CLASS</td>
<td>When a custom CSS file is assigned to the APPLICATION component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>IMAGE</td>
<td>This is the location of the main image file.</td>
</tr>
<tr>
<td>HOVER IMAGE</td>
<td>This is the location of the image file that shows when the user hovers over the image.</td>
</tr>
<tr>
<td>CLICK IMAGE</td>
<td>This is the location of the image file that is shown when the user clicks the image.</td>
</tr>
<tr>
<td>OPACITY %</td>
<td>By controlling this property, you can establish how transparent the IMAGE component is. 0% opacity means that the IMAGE component isn’t visible, and it becomes more opaque the closer to 100% you go. This property can be used for layout reasons; by setting the opacity of some IMAGE components a bit higher than others, you can send a subtle but clear message to the user. For example, if you have On and Off buttons, you can make the buttons that are in the On position more opaque than those in the Off position. This gives you the ability to convey a lot of information about the state of the buttons without using much room or color, thus keeping your design easy to grasp for the user.</td>
</tr>
<tr>
<td>TOOLTIP</td>
<td>This is the text message shown when the user hovers over the image.</td>
</tr>
<tr>
<td>ON CLICK</td>
<td>This opens the Script Editor to add user interaction.</td>
</tr>
</tbody>
</table>

Table 7.20  Image Component Properties

The IMAGE component can often come in handy, for example, when creating scorecards. For a scorecard, green, yellow, and red symbols are typically used. In Listing 7.6, the script evaluates the value of a key figure in the data source. If the value of the key figure is 1, then a green light image is used; if the value of the key figure is 2, then a yellow light will show; and for the value 3, a red light will show.

Instead of writing the location of the image in these statements, we’ve introduced three global variables holding the location of the images. The three variables are Greenlight, Yellowlight, and Redlight.

```javascript
Var EvalData=DS_1.getDataAsString("ZBB_NMBR","ZBB_KPIID=00001")
if (EvalData == "1")
  {IMG_FIN1_EVAL.setImage(Greenlight);}
else {
  if (EvalData == "2")
    {IMG_FIN1_EVAL.setImage(Yellowlight);}
  else {
    if (EvalData == "3")
      {IMG_FIN1_EVAL.setImage(Redlight);}
  } |

Listing 7.6  Script to Change Image Based on the Key Figure Value

Input Field Component

The INPUT FIELD component enables the user to type specific content into the application at runtime (Figure 7.59). For example, an INPUT FIELD component can be useful for filtering with wildcards (*). This kind of filtering is possible because the user is free to type anything he wants into the component’s text box.

Table 7.20 shows the properties for the INPUT FIELD component.
The Input Field component can come in quite handy, for example, if you don’t want to restrict a user in setting a filter value. For example, if a user wants to select all the years in the range 2001–2009, he can input "20*", and with that value, a filter can be set on the data source with this script:

```javascript
Var selectedYear = INPUTFIELD_1.getValue();
DSSALESYEAR.setFilterExt("0CALYEAR", selectedYear);
```

As another example, if you want to filter on products and you have a lot of different kinds of product types in your product line (red sauce, green sauce, sweet sauce, etc.), a user can input "*sauce" and apply the filter to see which products come up.

This also could set up a cascading filter where a List Box or Dropdown Box component is filtered based on the input in the Input Field component.

### List Box Component

The List Box component enables users to select items (Figure 7.60). A selected item can be used to filter for or choose a particular functionality in the application. The List Box component shows all the values in a list, so it’s advisable to limit the number of items.

### List Box Component versus Dropdown Box Component

If the number of items is large, a Dropdown Box component is more advisable because the list of values is collapsed when the user isn’t selecting a value.

---

Table 7.20 Input Field Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS CLASS</td>
<td>When a custom CSS file is assigned to the APPLICATION component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>EDITABLE</td>
<td>This specifies whether the field is editable.</td>
</tr>
<tr>
<td>TOOLTIP</td>
<td>This is the text message that will be shown when a user hovers the mouse pointer over the component.</td>
</tr>
<tr>
<td>VALUE</td>
<td>This is the initial value of the INPUT FIELD component. The VALUE can be set initially manually, via script, or via data binding. The user can change the value at runtime.</td>
</tr>
<tr>
<td>ON CHANGE</td>
<td>This opens the Script Editor to add user interaction.</td>
</tr>
</tbody>
</table>

Table 7.21 shows the properties of the List Box component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS CLASS</td>
<td>When a custom CSS file is assigned to the APPLICATION component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>ITEMS</td>
<td>With this property, the items available in the List Box component can be edited. A key has to be entered for each value. Having a text and a default item is optional. Click the button on the right side of the property to open an EDIT SCREEN dialog box where you can add, edit, or remove items. Additionally, you can populate the ITEMS list using either script or data binding.</td>
</tr>
<tr>
<td>MULTIPLE SELECTION</td>
<td>This specifies whether the user is allowed to select multiple items.</td>
</tr>
<tr>
<td>ON SELECT</td>
<td>This opens the Script Editor to add user interaction.</td>
</tr>
</tbody>
</table>

Consider a case where a user has to navigate through a lot of data. In this situation, you probably want a way to drill down instead of finding your way through a lot of choices. With the help of List Box components, you can achieve this by creating one List Box component that contains all the product categories and a second List Box component that contains all the products. When you select a product category from the first List Box component, the second List Box component is populated with the products within that category. Using this technique, you can lead the user step-by-step through the possible choices.

Let’s look at an example where you apply a category filter to the data source. The data source repopulates the items of the second List Box component with the accompanying products and makes this second component visible, while the first List Box component is set to invisible. Finally, when the user selects a product on the second List Box component, the script in this component filters the data source on that product. Other components that use that same data source then only show the data for that one product.
The code is shown in Listing 7.7.

```
DS_1.setFilterExt("0PRODUCTGROUP", LISTBOX_CATEGORY.getSelectedValue());
Var ProductItems = DS_1.getMemberList("0PRODUCT", MemberPresentation.EXTERNAL_KEY, MemberDisplay.TEXT, 20);
LISTBOX_PRODUCT.setItems(ProductItems);
LISTBOX_CATEGORY.setVisible(false);
LISTBOX_PRODUCT.setVisible(true);
```

Listing 7.7 Script to Create a Drill Through

Instead of using script to filter the data source and populate the second List Box, you can use input and output binding on the Item property of the first List Box. In that case, the only script lines you need are the setVisible() script lines.

Radio Button Group Component

The RADIO BUTTON GROUP component enables users to select a single item from a list (Figure 7.61). A selected item can be used to filter or to make a choice for a particular functionality in the application. Because the RADIO BUTTON GROUP component shows all the values in a list, we recommend limiting the number of items to not overwhelm the user with too many options.

![Figure 7.61 Radio Button Group Component](image)

Table 7.22 shows the properties of the RADIO BUTTON GROUP component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS CLASS</td>
<td>When a custom CSS file is assigned to the APPLICATION component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>COLUMNS</td>
<td>This is the number of columns used to display the RADIO BUTTON GROUP components.</td>
</tr>
</tbody>
</table>

Table 7.22 Radio Button Group Component Properties

The RADIO BUTTON GROUP component should be used when you need to give users different navigational choices. For example, if you allow your sales team to navigate through the customer base, there are many ways they might segment their customer base. For instance, they may want to look at married customers in their fifties who bought something in the past four weeks and show a pattern of purchases that puts them in the luxury buyers segment. Using four checkbox groups allows this kind of selection (Figure 7.62).

![Figure 7.62 Selection Options with Radio Button Groups](image)

Two RADIO BUTTON GROUP components together with the five selection options give the user a variety of choices. Now imagine four RADIO BUTTON GROUP components, with each holding five options. This creates 625 different combinations for the user to choose from. In other words, this component offers a lot flexibility.

Text Component

The TEXT component is used to add text, such as labels or values, to an application (Figure 7.63). TEXT components can be formatted at runtime by reassigning the CSS class. With all the possibilities of CSS, the TEXT component is a very flexible tool to use in an application. With the DATA BINDING option, you can set texts at runtime without having to resort to the TEXT POOL component while still having all the flexibility and central management in place.
Components and Properties

Figure 7.63  Text Component Using CSS for Layout

Note
You can even set the width of the Text component using CSS, although this is also possible via the common properties of components.

Table 7.23 shows the properties of the Text component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS CLASS</td>
<td>When a custom CSS file is assigned to the Application component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>STYLE</td>
<td>This is the style applied to the Text component. In the application, the text will get a CSS class with standard formatting. For Header 1, it’s the class .sapUiTvH2, for Header 2, it’s class .sapUiTvH2, and so on. If you want to enhance these stylings you can update these classes in the custom CSS.</td>
</tr>
<tr>
<td>TOOLTIP</td>
<td>This is a text message that will appear when the user hovers the mouse over the Text component.</td>
</tr>
<tr>
<td>CSS STYLE</td>
<td>You can insert CSS code to further enhance the layout of the component. The starting point for the CSS code inserted here is based on the theme of the application and the assigned CSS class, so keep this in mind when setting this property. If the theme sets the font size a little higher, the class sets the font size a little higher, and you set the font size in this property a little higher, your font size might turn out to be enormous.</td>
</tr>
<tr>
<td>ON CLICK</td>
<td>This opens the Script Editor to add user interaction when the user clicks on the Text component.</td>
</tr>
</tbody>
</table>

Table 7.23  Text Component Properties

CSS Tip
When applying formatting using CSS, try to use the CSS CLASS property as much as possible. When a class is altered in the central CSS file, all dependent components will automatically be updated. When using the CSS STYLE, however, you only influence the components where you have put the CSS code. This means when the same code is used in more than one component, any changes to the layout require that you manually change each component.

Tree Component
The Tree component can be bound to a hierarchical dimension in a data source (Figure 7.64). If there is no hierarchy, the Tree component shows a standard list. The Tree component shows the actual result set. Changes in the Tree component affect results in a Crosstab component tied to the same data source. Changes in a Crosstab component affect the Tree menu.

This component can be used as a navigational element. The component has methods to retrieve the selected elements, and you can work with that to add navigations.

Tree items are an important property in Tree components. You have to bind this property to a data source from where it collects the items to build the component.

7.3.4 Container Component Properties
Container components are used to group components into meaningful groups. These grouped components can be managed together by manipulating the container component (also known as the parent component) and therefore making maintenance easier. Components inside a container component are dependent on that container component; for example, if the container component’s visibility is set to False, then this setting will also apply to all the components inside the container component.
The margin properties are set in relation to the borders of their container component. For example, when the **Left Margin** of a container component is set to 100, and the **Left Margin** of a Text component within that container component is set to 10, this will set the text box at 110 pixels from the left side of the screen.

### Using Panel Components

Use Panel components in the application in the same way as you use folders on your computer. The Outline view will look like Windows Explorer/Finder with a hierarchical view of the components. Additionally, if you use meaningful names, it will be a lot easier to find the component you’re looking for in larger applications.

That being said, it’s possible to place components outside their container component but still maintain dependency. For example, you can set the top-, bottom-, left-, or right margin to a negative value. If you move the container component, the component outside will move along with the container but remain outside. In this case, if the **Left Margin** is set to –50, the object will be placed to the left of the container component, but there will still be a dependency between them.

In the example in Figure 7.65, you see a main component with a **Left Margin** of 25, a container component with a **Left Margin** of 50, and another container component with a **Left Margin** of 100. The total **Left Margin** for the main component will then be 175.

In the example in Figure 7.65, you see a main component with a **Left Margin** of 25, a container component with a **Left Margin** of 50, and another container component with a **Left Margin** of 100. The total **Left Margin** for the main component will then be 175.

![Figure 7.65 Layered Container Components with their Respective Left Margins](image)

There are five types of container components, all of which we discuss next.

### Grid Layout Component

The Grid Layout component is used to group and order the components in a grid (Figure 7.66). (The grid itself isn’t shown at runtime.) The Grid Layout component is very useful for dividing the screen into rows and columns. The sizes of the rows and columns can be adjusted by setting the relative size of each column and row.

![Figure 7.66 Grid Layout Component for Aligning Form Fields](image)

Table 7.24 shows the properties of the Grid Layout component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the component must be a unique name within the application.</td>
</tr>
<tr>
<td>CSS Class</td>
<td>Set the CSS class corresponding to a class defined in the custom CSS file to enhance the look of the grid. Because the component at runtime is an HTML table, you’ll find <strong>&lt;TR&gt;</strong> and <strong>&lt;TD&gt;</strong> elements that you can format using a class with a child selector (more in-depth CSS information is available in Chapter 10).</td>
</tr>
<tr>
<td>Number of Rows</td>
<td>This is the number of rows in the grid.</td>
</tr>
<tr>
<td>Row Height</td>
<td>This is the relative height of the row in comparison to the other rows. Each row starts with the value 1. Setting the rows' heights then divides the height of the grid according to the Row Height settings for each row.</td>
</tr>
<tr>
<td>Number of Columns</td>
<td>This is the number of columns in the grid.</td>
</tr>
<tr>
<td>Column Width</td>
<td>This is the relative width of the column in comparison to the other columns. Each column starts with the value 1. Setting the columns' widths divides the width of the grid according to the Column Width setting for each column.</td>
</tr>
</tbody>
</table>
You can create advanced layouts using several Grid Layout components. For example, if you have a main Grid Layout component with three rows, of which the middle row is the largest, you can divide the top and bottom row by adding a new Grid Layout component in the top row and bottom row.

Let’s consider a case where you choose to divide the top row into three columns and the bottom row into five. In Figure 7.67, you can see how this layout would look to the user. In Figure 7.68, you can see the outline as it would look in Design Studio at design time.

**Figure 7.67** Grid Layout View with Three Rows and Several Columns

![Grid Layout View with Three Rows and Several Columns](image)

**Figure 7.68** Outline View with Nested Grid Layout Components

![Outline View with Nested Grid Layout Components](image)

### Pagebook Component

The Pagebook component shows one page at a time and enables the user to switch to other pages by either swiping (iPad or iPhone) or dragging (computer mouse) (Figure 7.69). It’s also possible to switch pages by using the script language.

**Figure 7.69** Pagebook Component in the Outline View

![Pagebook Component in the Outline View](image)

In Figure 7.69, you can see several pages connected to the Pagebook component. In design mode, it’s possible to add or remove pages from the component. Each page is an empty canvas. Only the canvas of the selected page is visible to the user.

Table 7.25 shows the properties of the Pagebook component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CSS Class</strong></td>
<td>When a custom CSS file is assigned to the Application component, you can assign a CSS class here.</td>
</tr>
<tr>
<td><strong>Selected Page Index</strong></td>
<td>This sets the initial visible page of the Pagebook component. Note: 0 represents the first page of the Pagebook component, 1 represents the second page, and so on.</td>
</tr>
<tr>
<td><strong>Transition Effect</strong></td>
<td>This specifies the transition effect when a user swipes between pages. The following options are available: Slide In, Fade, Flip, Cube</td>
</tr>
<tr>
<td><strong>Transition Direction</strong></td>
<td>This specifies the transition direction when a user switches between pages. Options are Horizontal or Vertical.</td>
</tr>
</tbody>
</table>

**Table 7.25** Pagebook Component Properties
The Pagebook component is useful when you're designing menus and want an extra effect when changing pages. In Figure 7.70, you can see three menu items on the left and the data that corresponds to the second menu item on the right. The application is made in such a way that when you tap one of the menu items, the content on the right side will fade away, and new content relating to the menu choice will fade in. The fading effect is achieved by using the Transition Effect property, which you can set to Fade. The script in the text box contains the following line for the second menu:

```javascript
PAGEBOOK_1.setSelectedPageIndex(1);
```

This will result in the current screen fading away and being replaced with another screen.

An alternative way to do the same is addressing the page by name. The advantage is an improved readability of the script. In this example, we add an external parameter. If this isn’t set, the application will start with Page Main; otherwise, it uses the parameter that was passed to the application:

```javascript
var selectedPage = XPage;
if(selectedPage == "none") selectedPage = "MAIN";
PAGEBOOK_DASHBOARDS.setSelectedPageByName(selectedPage);
```

Numbering for Pagebook Pages
Although we mentioned it in Table 7.25, we want to emphasize this fact as it sometimes creates confusion: The numbering of PAGEBOOK pages starts at 0, and the second page is actually listed as page 1.

Panel Component
A PANEL component is used to group components together. By using the methods of the PANEL component, including .setCSSClass, it's possible to build interactivity into the application. As the number of components tends to grow quickly in an application, it's advisable to use these PANEL components often to group components that belong together, even if no interactivity is planned. Grouping allows you, for example, to hide a group of components at design time so your screen isn't cluttered when you're working on another part of the application.

Table 7.25 shows the properties of the PANEL component.
Components and Properties

7

290

Popup Component

The **Popup** component can be used to let a screen appear in the application, on top of all the other components, where users can make quick entries or configurations, see important messages, or make selections (Figure 7.71). **Popup** components can also be quite helpful for help messages and for providing further information about elements. The main reason to use a **Popup** component is to allow the user to perform a particular task and freeze the rest of the application until that task is done. You can ensure this by using the **Modal** property of the component.

**Figure 7.71** Example Usage for a Popup Component

There are two restrictions you should consider when using a **Popup** component:

- The **Popup** component can only be located in the root layout and not within another container element.
- The **Popup** component can only be positioned in relation to the root layout.

Table 7.27 shows the properties of the **Popup** component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CSS CLASS</strong></td>
<td>When a custom CSS file is assigned to the application component, you can assign a CSS class here.</td>
</tr>
<tr>
<td><strong>CSS STYLE</strong></td>
<td>In this setting, enter the CSS code to change the layout of the <strong>Panel</strong> component.</td>
</tr>
<tr>
<td><strong>ON CLICK</strong></td>
<td>This opens the Script Editor.</td>
</tr>
</tbody>
</table>

Table 7.26 Panel Component Properties

**Table 7.27** Popup Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NAME</strong></td>
<td>The name of the component must be a unique name within the application.</td>
</tr>
</tbody>
</table>

Table 7.27 Popup Component Properties (Cont.)

- **MODAL**
  - If the **POPUP** component is set to **MODAL**, the user can only navigate within the popup screen. If the **MODAL** property is set to **FALSE**, the user can also interact with other elements of the application.

  - **AUTOCLOSE**
    - This specifies whether the popup screen automatically closes when the user interacts outside of the popup in the application.

  - **ANIMATION**
    - This specifies the animation effect when the popup is opened or closed:
      - **NO ANIMATION**
      - **FLIP ANIMATION**
      - **POP ANIMATION**
      - **HORIZONTAL SLIDE ANIMATION**
      - **VERTICAL SLIDE ANIMATION**

Table 7.27 Popup Component Properties (Cont.)

A useful way to work with a **Popup** component is to set up a **Button** component in the application and have the **Popup** component appear near the button. For example, if you want to show a **Popup** component with some application settings, a **Button** component to the top right of the screen can be combined with a **Popup** component. The setting for animation can be set to **Vertical Slide Animation**.

When you use the **Popup** component this way, it will look like the popup appears directly out of the button and immediately grabs the user’s attention, as the user was already clicking that button.

To show the **Popup** component, add this code in the first **Button** component:

```javascript
POPUP_CONFIGSCREEN.show();
```

You add a second **Button** component in the **Popup** component itself and set the following code in the second button component to hide the component again:

```javascript
POPUP_CONFIGSCREEN.hide();
```

Tabstrip Component

A **Tabstrip** component allows you to group your application into tabs (Figure 7.72). It’s an easy way to divide your application in multiple screens. By clicking a tab, the user can move to a different screen in the application. The **Tabstrip**
component works much the same way as the PAGEBOOK component, creating several tabs, only one of which is visible at a time. The difference with a PAGEBOOK component is the way it allows you to navigate. With the TABSTRIP component, you automatically have a direct way to go from one tab to any of the other tabs. With a PAGEBOOK component, you can only go to the previous or next page, or you have to write scripts and add BUTTON components to create this same functionality.

Figure 7.72 Tabstrip Component

Table 7.28 shows the properties of the TABSTRIP component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS CLASS</td>
<td>When a custom CSS file is assigned to the APPLICATION component, you can assign a CSS class here.</td>
</tr>
<tr>
<td>SELECTED TAB INDEX</td>
<td>This specifies the tab that will initially be shown at the start. Index 0 opens the first tab of the component, index 1 the second, and so on.</td>
</tr>
<tr>
<td>ON SELECT</td>
<td>This opens the Script Editor. ON SELECT triggers each time a user selects a tab.</td>
</tr>
</tbody>
</table>

Table 7.28 Tabstrip Component Properties

If you want to go through the tabs step-by-step, you can add a BUTTON component outside the TABSTRIP. Each time the BUTTON component is clicked, the user goes to the next page until reaching the last (Listing 7.8). Then the TABSTRIP component jumps back to the starting tab.

Listing 7.8 Loop Through Tabs Using a Button Component

```javascript
var tabnumber = TABSTRIP_1.getSelectedTabIndex();
if (tabnumber == 4) {
  TABSTRIP_1.setSelectedTabIndex(0);
} else {
  TABSTRIP_1.setSelectedTabIndex(tabnumber + 1);
}
```

You could use this for a wizard-like navigation through several steps, and if the final step is done, return to the starting page. A wizard-like solution is handy when you want to enable users to do complicated tasks and guide them through the process.

Split Cell Container Component

The SPLIT CELL CONTAINER is an area where a user can drag and arrange portable fragments. The fragments are shown in a tabular format. The area is similar to a placeholder area where the user has his own room to arrange fragments.

Table 7.29 lists the properties of the SPLIT CELL CONTAINER component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON DROP</td>
<td>The event script for when a user adds a new fragment to the SPLIT CELL CONTAINER.</td>
</tr>
<tr>
<td>ON DELETE</td>
<td>The event script for when a user removes a fragment from the SPLIT CELL CONTAINER.</td>
</tr>
</tbody>
</table>

Table 7.29 Split Cell Container Properties

7.4 Technical Component Properties

In this section, we’ll discuss the properties of all the technical components. You won’t find these in the COMPONENTS pane, but you can add them directly in the OUTLINE pane in the TECHNICAL COMPONENTS folder. These kinds of components play a central role in the application and are used in relation with other components.
## Contents

Foreword ......................................................................................................... 17  
Introduction ..................................................................................................... 19  
Acknowledgments ............................................................................................ 25  

**PART I Getting Started**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction to SAP BusinessObjects Design Studio</td>
<td>29</td>
</tr>
<tr>
<td>1.1</td>
<td>What Is SAP BusinessObjects Design Studio?</td>
<td>30</td>
</tr>
<tr>
<td>1.1.1</td>
<td>Development Environment</td>
<td>32</td>
</tr>
<tr>
<td>1.1.2</td>
<td>Components</td>
<td>32</td>
</tr>
<tr>
<td>1.1.3</td>
<td>Data Sources and Platforms</td>
<td>33</td>
</tr>
<tr>
<td>1.1.4</td>
<td>Scripting</td>
<td>34</td>
</tr>
<tr>
<td>1.1.5</td>
<td>Cascading Style Sheets</td>
<td>34</td>
</tr>
<tr>
<td>1.1.6</td>
<td>Templates and Themes</td>
<td>34</td>
</tr>
<tr>
<td>1.1.7</td>
<td>Application Execution</td>
<td>35</td>
</tr>
<tr>
<td>1.2</td>
<td>SAP BusinessObjects Design Studio and Existing SAP BusinessObjects BI Environments</td>
<td>37</td>
</tr>
<tr>
<td>1.2.1</td>
<td>SAP BusinessObjects BI Environments</td>
<td>38</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Content Creation and Consumption</td>
<td>44</td>
</tr>
<tr>
<td>1.3</td>
<td>Recent Developments</td>
<td>46</td>
</tr>
<tr>
<td>1.4</td>
<td>Summary</td>
<td>48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>SAP BusinessObjects Design Studio vs. SAP BusinessObjects Dashboards vs. BEx Web Application Designer</td>
<td>49</td>
</tr>
<tr>
<td>2.1</td>
<td>SAP BusinessObjects Dashboards</td>
<td>49</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Setting Up a Dashboard</td>
<td>51</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Components</td>
<td>53</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Data Connectivity</td>
<td>56</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Publishing</td>
<td>57</td>
</tr>
<tr>
<td>2.1.5</td>
<td>SDK</td>
<td>57</td>
</tr>
<tr>
<td>2.2</td>
<td>BEx Web Application Designer</td>
<td>58</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Setting Up a Web Application Template</td>
<td>60</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Web Items</td>
<td>62</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Publishing</td>
<td>64</td>
</tr>
</tbody>
</table>
## Contents

### 2. Key Differences
- 2.3 Key Differences ................................................. 64
- 2.4 Summary ....................................................... 68

### 3. Usage Scenarios .................................................... 69
- 3.1 Real-Time Production Dashboard ......................... 69
- 3.2 Sales Dashboard Application ................................. 75
  - 3.2.1 Main Screen ............................................. 75
  - 3.2.2 Detail Screen ........................................... 77
  - 3.2.3 Quotes .................................................. 79
- 3.3 OLAP Application .............................................. 81
  - 3.3.1 Starting the Application .............................. 82
  - 3.3.2 Customizing the Application ......................... 83
- 3.4 Summary ....................................................... 89

### 4. Installation and Configuration ................................. 91
- 4.1 Architecture, Components, and Prerequisites .......... 92
  - 4.1.1 SAP BusinessObjects BI Platform .................. 94
  - 4.1.2 SAP Business Warehouse ............................. 95
  - 4.1.3 SAP Enterprise Portal ................................. 95
  - 4.1.4 SAP HANA .............................................. 95
  - 4.1.5 SAP BusinessObjects Design Studio Client Tool . 96
  - 4.1.6 Browsers ................................................ 96
- 4.2 Preparing for Installation ..................................... 96
  - 4.2.1 Accessing Document Guides ......................... 96
  - 4.2.2 Downloading Software Components ................. 97
  - 4.2.3 Helpful SAP Notes .................................... 98
  - 4.2.4 Extracting Installation Files ......................... 98
- 4.3 Installing the SAP BusinessObjects BI Platform Add-On 100
- 4.4 Configuring the SAP BusinessObjects BI Platform .......... 108
  - 4.4.1 Initializing the Analysis Application Service .... 109
  - 4.4.2 Setting the Number of Client Sessions .......... 113
  - 4.4.3 Assigning User Authorizations ................... 114
  - 4.4.4 Creating a Mobile Category ....................... 118
  - 4.4.5 Creating an SAP BW OLAP Connection ......... 120
  - 4.4.6 Creating an SAP HANA OLAP Connection ..... 123
- 4.5 Configuring SAP Enterprise Portal and SAP Business Warehouse 124
- 4.6 Configuring SAP HANA ....................................... 131

### 5. The Integrated Development Environment .................. 143

#### 5.1 Menu ........................................................... 143
  - 5.1.1 Application ............................................... 143
  - 5.1.2 Planning ................................................. 155
  - 5.1.3 Edit ..................................................... 156
  - 5.1.4 Layout .................................................. 157
  - 5.1.5 Search .................................................. 158
  - 5.1.6 View ................................................... 160
  - 5.1.7 Tools ................................................... 162
  - 5.1.8 Help .................................................... 175
- 5.2 Toolbar .......................................................... 177
  - 5.2.1 Toolbar .................................................. 177
  - 5.2.2 Toolbar .................................................. 177
  - 5.2.3 Toolbar .................................................. 177
- 5.3 Layout Editor ................................................... 179
  - 5.3.1 Components View .................................... 180
  - 5.3.2 Outline View .......................................... 181
  - 5.3.3 Properties View ...................................... 186
  - 5.3.4 Additional Properties View ....................... 190
  - 5.3.5 Error Log View ...................................... 192
  - 5.3.6 Script Problems View ............................... 193
- 5.4 Summary ....................................................... 193

### 6. The Application Design Process ............................... 195

#### 6.1 Setting Up the User Interface and Visualizations .......... 197
  - 6.1.1 Setting Up the User Interface ...................... 197
  - 6.1.2 Visualizations ........................................ 202
- 6.2 Adding the Data .............................................. 208
- 6.3 Making It Interactive ....................................... 208
- 6.4 Formatting and Fine-Tuning ............................... 211
- 6.5 Executing the Application .................................. 214
- 6.6 Summary ....................................................... 216

### 7. Configuring Connections via Universes and
Custom Data Sources .................................................. 133
- 7.1 Universes ..................................................... 133
  - 7.1.1 Configuring Connections via Universes ........ 133
  - 7.1.2 Custom Data Sources ............................... 134
- 7.2 Installing the SAP BusinessObjects Design Studio Client Tool 135
- 7.3 Summary ....................................................... 140
## PART IV The Software Development Kit

### 12 Using SDK Extensions in SAP BusinessObjects Design Studio 469

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1 Extensions: What They Are and How They Work</td>
<td>469</td>
</tr>
<tr>
<td>12.2 Installing, Upgrading, and Removing SDK Extensions</td>
<td>471</td>
</tr>
<tr>
<td>12.2.1 Installing</td>
<td>471</td>
</tr>
<tr>
<td>12.2.2 Upgrading</td>
<td>478</td>
</tr>
<tr>
<td>12.2.3 Uninstalling the SDK</td>
<td>480</td>
</tr>
<tr>
<td>12.2.4 Moving an SDK to a Lower SDK Version</td>
<td>481</td>
</tr>
<tr>
<td>12.3 Moving an SDK through the System Landscape</td>
<td>482</td>
</tr>
<tr>
<td>12.3.1 Promoting in SAP BusinessObjects BI</td>
<td>483</td>
</tr>
<tr>
<td>12.3.2 Transporting with SAP Business Warehouse</td>
<td>483</td>
</tr>
<tr>
<td>12.3.3 Transporting with SAP HANA</td>
<td>484</td>
</tr>
<tr>
<td>12.4 Testing Changes to an SDK Extension</td>
<td>484</td>
</tr>
<tr>
<td>12.5 Third-Party SDK Extensions</td>
<td>485</td>
</tr>
<tr>
<td>12.6 Summary</td>
<td>489</td>
</tr>
</tbody>
</table>

### 13 SDK Installation and Deployment 491

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1 Installing Eclipse</td>
<td>491</td>
</tr>
<tr>
<td>13.1.1 Java Development Kit</td>
<td>491</td>
</tr>
<tr>
<td>13.1.2 Eclipse</td>
<td>494</td>
</tr>
<tr>
<td>13.2 Registering the XML Definition</td>
<td>498</td>
</tr>
<tr>
<td>13.2.1 Downloading SAP Templates</td>
<td>498</td>
</tr>
<tr>
<td>13.2.2 Setting Up XML</td>
<td>499</td>
</tr>
<tr>
<td>13.3 Importing the Project</td>
<td>500</td>
</tr>
<tr>
<td>13.4 Setting the Target Platform</td>
<td>502</td>
</tr>
<tr>
<td>13.5 Summary</td>
<td>506</td>
</tr>
</tbody>
</table>

### 14 Building Components Using the SDK 507

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1 Understanding the SDK Framework</td>
<td>507</td>
</tr>
<tr>
<td>14.2 Languages Used in an SDK</td>
<td>509</td>
</tr>
<tr>
<td>14.2.1 JavaScript Overview</td>
<td>509</td>
</tr>
<tr>
<td>14.2.2 XML Overview</td>
<td>524</td>
</tr>
<tr>
<td>14.2.3 HTML Overview</td>
<td>525</td>
</tr>
<tr>
<td>14.2.4 CSS Overview</td>
<td>526</td>
</tr>
<tr>
<td>14.3 Building Blocks of an SDK Extension</td>
<td>527</td>
</tr>
<tr>
<td>14.3.1 Component.xml</td>
<td>527</td>
</tr>
<tr>
<td>14.3.2 Component JavaScript</td>
<td>528</td>
</tr>
</tbody>
</table>
### PART V  Examples

#### 15 Building a Real-Time Production Dashboard ................................. 617

15.1 Application Overview ......................................................... 617
15.2 Building the Application ...................................................... 618
15.2.1 Views ........................................................................ 620
15.2.2 CSS Classes ............................................................... 622
15.2.3 Components and Layout .............................................. 623
15.2.4 Adding Data to the Views .............................................. 628
15.2.5 Interaction ................................................................. 630
15.2.6 Applying Filters at Startup ............................................. 633
15.2.7 Timer Component ....................................................... 636
15.3 Summary ........................................................................... 638

#### 16 Building a Sales Dashboard Application ......................................... 639

16.1 Application Overview ......................................................... 639
16.2 Building the Application ...................................................... 642
16.2.1 Main Page ................................................................... 642
16.2.2 Detail Screen ............................................................... 660
16.2.3 Quotes ........................................................................ 666
16.3 Summary ........................................................................... 674

#### 17 Building an OLAP Application Using a Template .......................... 675

17.1 Creating the Application ...................................................... 675
17.2 Application Properties ....................................................... 682
17.3 Application Components .................................................... 685
17.3.1 Header ....................................................................... 687
17.3.2 Body ......................................................................... 688
17.3.3 Buttons ........................................................................ 690
17.3.4 On Startup ................................................................. 695
17.3.5 Adding Filters ............................................................. 697
17.4 Summary ........................................................................... 698

#### 18 Outlook ............................................................................. 709

18.1 SAP Analytics and Platform Strategy ..................................... 699
18.2 Roadmap ................................................................. 702
18.2.1 Planned Innovations .................................................. 704
18.2.2 Future Direction ......................................................... 705
18.3 Summary ........................................................................... 705

### Appendices ........................................................................ 715

B SAP BusinessObjects Mobile and SAP BusinessObjects Design Studio ............................................ 715
C The Authors ........................................................................ 723

Index ....................................................................................... 725
## Index

<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>.!important</td>
<td>687</td>
</tr>
<tr>
<td>.concat</td>
<td>512</td>
</tr>
<tr>
<td>.exit()</td>
<td>564</td>
</tr>
<tr>
<td>.getselectedvalue</td>
<td>306</td>
</tr>
<tr>
<td>.save</td>
<td>394</td>
</tr>
<tr>
<td>.splice</td>
<td>513</td>
</tr>
<tr>
<td>@media</td>
<td>415</td>
</tr>
<tr>
<td>100% stacked bar chart</td>
<td>439</td>
</tr>
<tr>
<td>100% stacked column</td>
<td>441</td>
</tr>
<tr>
<td>Action Sheet component</td>
<td>301</td>
</tr>
<tr>
<td>activateHierarchy</td>
<td>325</td>
</tr>
<tr>
<td>Active Provider</td>
<td>171</td>
</tr>
<tr>
<td>Ad hoc</td>
<td>111</td>
</tr>
<tr>
<td>Adaptive Processing Server</td>
<td>109, 477</td>
</tr>
<tr>
<td>Ad hoc</td>
<td>111</td>
</tr>
<tr>
<td>Additional properties</td>
<td>508, 529, 530, 607</td>
</tr>
<tr>
<td>afterDesignStudioUpdate</td>
<td>572</td>
</tr>
<tr>
<td>afterUpdate</td>
<td>545, 547, 549, 562, 596</td>
</tr>
<tr>
<td>alert</td>
<td>522</td>
</tr>
<tr>
<td>Analysis Application Design Service</td>
<td>305</td>
</tr>
<tr>
<td>Analysis Application Service</td>
<td>105, 109</td>
</tr>
<tr>
<td>Analysis tab</td>
<td>82, 676</td>
</tr>
<tr>
<td>Analytic component</td>
<td>242</td>
</tr>
<tr>
<td>Analytic view</td>
<td>123</td>
</tr>
<tr>
<td>Android</td>
<td>715</td>
</tr>
<tr>
<td>Anonymous functions</td>
<td>565</td>
</tr>
<tr>
<td>Anscombe’s quartet</td>
<td>435</td>
</tr>
<tr>
<td>api</td>
<td>559, 571</td>
</tr>
<tr>
<td>ApplicationInfo</td>
<td>693</td>
</tr>
<tr>
<td>Application</td>
<td>29, 92</td>
</tr>
<tr>
<td>add data source</td>
<td>150</td>
</tr>
<tr>
<td>building principles</td>
<td>453</td>
</tr>
<tr>
<td>building tips</td>
<td>454</td>
</tr>
<tr>
<td>close</td>
<td>149</td>
</tr>
<tr>
<td>complex</td>
<td>453</td>
</tr>
<tr>
<td>Application (Cont.)</td>
<td>149</td>
</tr>
<tr>
<td>create new</td>
<td>145, 197</td>
</tr>
<tr>
<td>delete</td>
<td>149</td>
</tr>
<tr>
<td>execute locally</td>
<td>153, 214</td>
</tr>
<tr>
<td>execute on SAP BusinessObjects BI</td>
<td>154</td>
</tr>
<tr>
<td>exit</td>
<td>155</td>
</tr>
<tr>
<td>open</td>
<td>148</td>
</tr>
<tr>
<td>open recent</td>
<td>148</td>
</tr>
<tr>
<td>output</td>
<td>361</td>
</tr>
<tr>
<td>preferred startup mode</td>
<td>155</td>
</tr>
<tr>
<td>recovery</td>
<td>165</td>
</tr>
<tr>
<td>responsive</td>
<td>414</td>
</tr>
<tr>
<td>save</td>
<td>149</td>
</tr>
<tr>
<td>size</td>
<td>455</td>
</tr>
<tr>
<td>template</td>
<td>167</td>
</tr>
<tr>
<td>title</td>
<td>624</td>
</tr>
<tr>
<td>Application component</td>
<td>329, 356</td>
</tr>
<tr>
<td>custom CSS</td>
<td>218, 220</td>
</tr>
<tr>
<td>displayed message types</td>
<td>219</td>
</tr>
<tr>
<td>force prompts on startup</td>
<td>219</td>
</tr>
<tr>
<td>global script variables</td>
<td>219</td>
</tr>
<tr>
<td>On Startup</td>
<td>220</td>
</tr>
<tr>
<td>position of message button</td>
<td>218</td>
</tr>
<tr>
<td>position of message window</td>
<td>218</td>
</tr>
<tr>
<td>properties</td>
<td>218</td>
</tr>
<tr>
<td>Application design process</td>
<td>195, 196</td>
</tr>
<tr>
<td>adding data</td>
<td>202</td>
</tr>
<tr>
<td>executing the app</td>
<td>214</td>
</tr>
<tr>
<td>formatting</td>
<td>211</td>
</tr>
<tr>
<td>interactivity</td>
<td>208</td>
</tr>
<tr>
<td>UI and visualizations</td>
<td>197</td>
</tr>
<tr>
<td>ApplicationInfo Object Component</td>
<td>331</td>
</tr>
<tr>
<td>Architecture</td>
<td>92</td>
</tr>
<tr>
<td>Archius</td>
<td>487</td>
</tr>
<tr>
<td>Area chart</td>
<td>442</td>
</tr>
<tr>
<td>Arguments</td>
<td>307</td>
</tr>
<tr>
<td>Array</td>
<td>315, 512, 539, 554</td>
</tr>
<tr>
<td>elements</td>
<td>512</td>
</tr>
<tr>
<td>JSON object</td>
<td>514</td>
</tr>
<tr>
<td>length</td>
<td>513</td>
</tr>
<tr>
<td>nested</td>
<td>513</td>
</tr>
<tr>
<td>object</td>
<td>512</td>
</tr>
<tr>
<td>Assignment statements</td>
<td>306, 308</td>
</tr>
<tr>
<td>attachChange</td>
<td>572</td>
</tr>
</tbody>
</table>
Component (Cont.)
On Startup, 225
reload, 154
show, 185
Component CSS, 531, 584
Component JavaScript, 528, 544, 550, 555, 571, 577, 578, 610
adding functions, 546
create, 592
outline, 592
componentDeleted, 545, 549
Components
Internal formatting, 462
types, 311
Concatenates, 309
Conceptual rules, 451
Condition, 521
Conditional formatting, 237, 249, 325
Conditional statements, 306, 307, 519
Configuration, 91
Connection, 685
Constructor, 516
Container component
properties, 283
Content Assistant, 313, 314, 531
Context menu
enabled, 248
Context menu component, 335
contribution.xml, 496, 509, 527, 531, 549, 575
contribution.ztl, 531, 558
Convert component, 323
copyFilter, 393
copyVariableValue, 393
Counter object, 523
Create
child, 679
new, 175
Crosstab component, 84, 247, 390, 426
column limit, 248
CSS class, 248
data source, 248
OLAP application, 83
pixel-based scrolling, 248
properties, 248
row limit, 248
CSS, 34, 218, 305, 399, 401, 414, 581
additional properties, 239
advanced, 413
attribute selectors, 526
basic selectors, 526
browsers, 414
class, 253, 257, 266, 272, 336, 404, 413, 532, 622, 658
color codes, 407
custom classes, 408
font size, 407
group classes, 414
hiding panels, 416
include, 537
layout, 461
loading fonts, 657
style, 282, 404
style properties, 462
themes, 401
CSS Style Editor, 405, 410
CUID, 372
Currency table, 703
Custom
data sources, 133, 134
Custom Chart component, 77
Custom measures, 703
Customer feedback, 640
D
D3, 507, 559, 561, 601, 602, 604
adding columns and rows, 563
adding elements, 561
data binding, 563
functions, 599
library, 518, 537, 561
scales, 569
transitions, 566
D3.selectAll, 594
Dashboards, 29, 30, 50
Data
binding, 52, 262, 541
bound, 536, 540, 555, 586, 587
cell binding, 262
discovery, 147
grouping, 607
manipulation, 359
Data (Cont.)
preparation, 361
selection, 629
series, 239
visualization, 29, 147
Data Field component, 271
Data Runtime JSON, 549, 553, 554
Data source, 152, 169, 181, 202, 328, 464, 679
add, 150, 203
browser, 456
manually entering records, 650
pause refresh, 183
show prompts, 311
Drilldown report, 639
DS_1, 682
Dynamic properties, 568
E
Eclipse, 31, 491, 494, 557
application, 503
environment, 495
installation, 487, 491
Edit menu, 156
copy, 157
delete, 157
paste, 157
redo, 156
undo, 156
Elements, 512, 524, 559
enter(), 564, 568, 599
Enterprise assets, 700
Enum, 311
Event handler, 312
Events, 684
Execute, 393, 394
locally, 153, 214
on SAP BusinessObjects BI platform, 154
exit, 568
Export as Template, 167
Expression, 309
types, 310
Expressions, 307
Extends, 558
Extension level, 533
External parameter, 225, 455
Fact data, 552
Fact Data Runtime, 552
Fake data, 573
Federated, 408
fillDropDown, 610
fillMetadataProperty, 543
Filter Line component, 146, 252
Filter Panel component, 256, 344
data source, 257
dimensions, 257
on apply, 257
properties, 256
target data source, 257
Filters, 85, 87, 255, 257, 325, 633, 636
area, 685
cascading, 654
values, 87
firepropertiesChanged, 602
Fixed structure, 639
flaticon, 659
Float, 310
number, 324
Font file, 657
font-weight, 408
Footer, 685
panel, 416
For each – for each – loop over array, 315
for statement, 521
Forecasts, 392
Formatted Text Field component, 273
Formatted Text View component, 693
Fragment Gallery component, 146, 273
Full time equivalent (FTE), 195
Function call, 544
Functions, 546, 557
Galigeo, 487
Generic Analysis template, 147, 677
Geo Map component, 258, 339, 703
properties, 259
GeoJSON file, 261
GeoJSON Mapping Property, 261
getDimensions, 557
getter/setter, 511, 547, 595
Getting started, 175
Global script object, 296, 321, 630
Global script variables, 225, 312, 634
GridLayout component, 285
column width, 285
name, 285
number of columns, 285
number of rows, 285
properties, 285
row height, 285
Guidelines, 453
handlerType, 571
hasClientChanges, 399
Header, 249, 624, 685, 687
panel, 416
Heat map, 681
Help menu, 175
Hierarchy, 182
Hierarchy node, 552
High Contrast Black theme, 401, 403
Highlight, 235
Highlight outliers, 443
Horizontal area chart, 442
Horizontal line chart, 437
Horizontal waterfall chart, 445
hover, 581
HTML, 509, 525, 608
HTML5, 31, 525
HTTP protocol, 170
IBCS, 432, 434, 451
Icon component, 274
properties, 275
ICON_GROUP_ANALYSIS, 690
ICON_PROMPTS, 694
icons, 532, 536, 585, 666
IDs, 525, 526
if – if statement, 316
if statement, 512
Image
file, 658
resizing, 659
Image component, 275, 346
CSS class, 276
image, 276
properties, 276
includeData, 544
increment, 521, 523
In-depth dashboard, 645
Index number, 513
Info Chart component, 240, 339
Info Chart Feeding Panel component, 241
InfoAreas, 380
InfoProvider, 341
InfoProviders, 33
Init(), 546, 561, 577, 594
Initial values, 591
Initialization, 521
Input and output parameters, 298
Input Field component, 277, 347
Input Field component, 277, 347
CSS class, 278
properties, 277
value, 278
input-ready, 327
Installation, 91, 96
Analysis Application Support for Mobile Services, 105
Analysis Application Web component, 104
document guides, 96
extract files, 98
full or custom, 104
software components, 97, 98
wizard, 102
Instance, 516
Integrated development environment (IDE), 143
Interactivity, 305
Inverse scale, 569
iView template, 130
Java, 493
uninstall, 493
Java Development Kit (JDK), 491
install, 491
Java Runtime Environment (JRE), 96
JavaScript, 34, 508, 509, 529, 609
classes, 516
conditional statements, 519
inheritance, 517
libraries, 559
methods, 517
object-oriented, 516
properties, 516
JavaScript Object Notation (JSON), 318
jQuery, 507, 559
library, 510, 518
manipulating elements, 560
selecting elements, 559
selections, 559
jsInclude, 531, 537
JSON, 310, 318, 320, 514, 548, 553, 556
dimension members, 556
JSON Grabber, 557
Key, 85
Key figures, 552
KPI, 43, 69, 72, 76, 573, 618, 640
quality, 618, 628
title, 622
Language support, 703
Layout Editor, 179
Layout menu, 157
Maximize Component, 158
align commands, 157
Language support, 703
Layout Editor, 179
Layout menu, 157
Maximize Component, 158
align commands, 157
Leverage investments, 700
Lead the cloud, 699
Language support, 703
Layout Editor, 179
Layout menu, 157
Maximize Component, 158
align commands, 157
Leverage investments, 700
Lifecycle Management Console (LCM), 93
Line chart, 378, 436
Linear scale, 569
List Box component, 278, 345, 363, 691
CSS class, 279
items, 279
on select, 279
properties, 279
vs. Dropdown Box component, 278
Load in script, 227, 228
Loading state indicator, 336
Loading, 409
Local mode, 139, 154
Local, 409
Local file installation, 478
Local mode, 139, 154
repository folder, 154
Local variables, 311, 511
Log level, 173
Logical name, 410
Loops, 521

Methods (Cont.)
ApplicationInfo object, 331
Bookmark component, 331
Button component, 337
Chart Component, 337
Checkbox component, 340
Checkbox Group component, 341
common, 335
create, 557
 Crosstab component, 342
Date Field component, 343
Dimension Filter component, 344
Filter Line component, 344
Filter Panel component, 344
Image component, 346
Input Field component, 347
Math component, 312
Navigation Panel component, 345
Pagebook component, 348
Panel component, 348
Popup component, 348
Scorecard component, 340
Spreadsheet component, 340
Tabstrip component, 349
Text component, 349
Microsoft Excel, 51, 709
MIME repository, 411
Mobile, 31, 35, 118, 703
category, 118
collection, 717
device, 177, 215
running Design Studio content, 718
server, 119, 716
theme, 402, 404
Modes, 536
Multidimensional arrays, 513
Multiple pie chart, 447
Multiple radar chart, 448
Multisource relational UNX, 134
myBodyPanel, 416
myButton, 631
myCheckBox, 422
myFooterPanel, 416
myTitle, 415

N
Naming conventions, 456, 458
Navigation, 257, 465
items, 350
menu, 350
popup, 353
Navigation Panel component, 345, 374
Network connections, 171
Nodes, 524
Notepad++, 410
Number of dimensions, 651

O
Object constructor, 516
ODBC Data Source Administrator, 170
OLAP application, 81, 373, 675
adding charts, 376
additional options, 88
bookmarking, 88, 691
chart options, 86
collection, 685
collectionId, 379
create, 675
dashboard, 82
design, 83
editing views, 83
export, 684
filters, 87, 697
layout, 678
navigation panel, 83
output, 85
properties, 682, 695
selecting a data source, 379
tabs, 82, 676
OLAP connection, 120, 123
define authentication method for SAP BW, 122
SAP HANA, 123
On Click, 632
On Startup, 681, 695
OnChange, 656
Online Analytical Processing (OLAP), 81
Online composition, 147
OnTimer, 637
Opacity, 276
Open data source browser, 691
OpenDocument, 214, 312
OpenOffice.org, 129
Ordinal scale, 570
Outline panel, 456
Overall Equipment Efficiency (OEE), 69, 70, 618
per plant, 71

P
Pagebook component, 287, 348, 375
CSS class, 287
page caching, 288
properties, 287
selected page index, 287
transition effect, 287
Panel component, 284, 289, 348, 456
CSS class, 290
CSS style, 290
properties, 289
Parameter, 546
Parent, 517
Part-to-whole relationship, 439
Performance, 70, 463
RPI, 618, 628
measurement, 330
Pie chart, 446
Planning, 683
applications, 147
connection, 219, 387
function, 387, 389, 392
functionality, 394
layout, 147
model, 219
object, 388
prerequisites, 385
sequence, 387, 389, 392, 393
Planning application
building, 395
Planning component, 333
Planning Function component, 333
Platinum theme, 402
Popup component, 146, 290, 348
animation, 291
autoclose, 291
model, 291
Index

Popup component (Cont.)
name, 290
possiblevalue, 540, 591
Predefined
statement, 315
statement templates, 317
Preferences menu, 163
application design, 163
scripting, 163
Primitive types, 310, 515
Product Availability Matrix (PAM), 94
Production, 70
Project
import, 499
importing, 500
Prompts, 152, 683
Properties, 217, 405, 516, 538
custom layout, 229
custom CSS, 220
description, 218
dome, 218
Properties panel, 534
Property
binding, 667
PropertyPage, 530
prototype.constructor, 518
Prototyping, 704
Purchasing dashboard, 82

Radio Button Group component (Cont.)
range, 280
properties, 280
Range, 254
Readability, 457, 460, 518
Ready-to-run, 147, 677
Realization column, 652
Real-time
view, 619, 633
Real-time production dashboard, 69, 617
components and layout, 623
interactivity, 630
Redundant selections, 257
References, 459
Regression test, 485
Relational UNIX, 134
Rendering mode, 146
Reports, 29
Reset, 395
ResultCellList, 553
returnData, 605
Reusable functions, 587
Right margin, 336
Row-based data models, 463
Rows, 83

S
S_RS_ZEN, 130
authorization fields, 130
Sales dashboard, 75, 639
class, 656
detail screen, 77
headers, 654
icon, 658
main page, 75, 642
PDF document, 79
quotes, 79
quotes page, 666
right side menu, 654
SAP Business Connectivity BI
promoting, 483
SAP Business Warehouse, 33, 92, 93, 95, 147,
170, 214, 411, 478, 491, 700, 703
SAP Business Warehouse BI
promoting, 483
SAP Crystal Reports, 37, 39, 700, 703, 715
SAP Crystal Reports, 37, 39, 700, 703, 715
SAP Event Stream Processing, 618
SAP Fiori, 146, 705
SAP HANA, 33, 66, 93, 95, 123, 147, 151,
170, 214, 478, 491, 700, 702, 705
SAP Business Warehouse (Cont.)
InfoArea, 151
InfoProvider, 150
Java Portal, 65
OLAP connection, 120
query view, 150
role, 151
setup for Design Studio, 124
supported versions, 95
transporting, 483
SAP BusinessObjects Analysis, 42
creating a Design Studio application, 710
smart copy, 709, 712
SAP BusinessObjects Analysis, edition for
Microsoft Office, 37, 40, 42, 81, 675, 700,
703, 709
SAP BusinessObjects Analysis, edition for
OLAP, 40, 42, 147, 709
SAP BusinessObjects BI, 33, 92, 93, 214, 379,
385, 411, 477, 700
add-on for Design Studio, 100
client session, 113
configuration for Design Studio, 108
log, 113
mobile server, 716
prerequisites, 94, 102
public folder, 223
strategy, 699
supported versions, 94
type, 310
user authorization, 114
SAP BusinessObjects BI tools
dashboarding and application creation, 43
discovery and analysis, 40
reporting, 38
SAP BusinessObjects Dashboards, 30, 37, 43,
704, 715
cart types, 53
categories, 53
customizer component, 55
data connectivity, 56
design environment, 52
default components, 52
example, 50
map, 55
publishing, 57
SDK, 57
selectors, 54
SAP BusinessObjects Dashboards (Cont.)
setup, 51
single-value component, 55
vs. Design Studio, 49, 64
SAP BusinessObjects Design Studio, 43
and SAP BusinessObjects BI, 37
BEx Application Designer and Dashboards, 37
client tool, 92, 93, 96, 135
components and properties, 217
configuration, 91
delivery form, 703, 705
installation, 91
mobile, 715
outlook and future developments, 699
planned innovations, 703, 704
roadmap, 702, 703
SAP BusinessObjects Mobile, 715
supported browsers, 96
usage scenario, 69
vs. BEx Web Application Designer, 49, 65
vs. SAP BusinessObjects Dashboards, 49, 64
SAP BusinessObjects Explorer, 37, 40, 705
SAP BusinessObjects Mobile, 53, 118, 715
annotation, 719
collaboration, 719
connectivity, 716
iOS, 715
supported platforms, 715
voice memo, 719
SAP BusinessObjects Web Intelligence, 37, 40,
700, 715
SAP BW Integrated Planning (SAP BW-IP), 64
SAP Consulting, 487
SAP Crystal Reports, 37, 39, 700, 703, 715
SAP Enterprise Portal, 93, 95, 130, 478
prerequisites, 95
setup for Design Studio, 124
SAP Event Stream Processing, 618

Q
QR code, 177
Quality, 70, 73
Quote
delivery form, 79
document, 642

R
Radar chart, 448
Radio Button component, 345
Radio Button Group component, 280
columns, 280
css class, 280
traits, 281
on select, 281

Index

734

735
Velocity, 704
Vendor, 533
View, 620
  additional properties, 190
  properties, 186
  script problems, 193
View menu, 160, 180
  additional properties, 161
  components, 161, 180
  error log, 162, 192
  outline, 161, 181
  properties, 161, 187
  reset layout, 162
  script problems, 161
  search results, 162
Visual components, 229, 335
Visualization, 429, 603
Web Application Container Services (WACS), 107
Welcome page, 139, 175
What-if scenario, 51
while-statement, 521
Width, 336
Word cloud graph, 78, 661
data source, 661
Workspace, 502
WYSIWYG, 32, 143, 179

X

Xcelsius, 51
XML, 524, 537, 567
catalog, 499
definition, 495
elements, 524
nodes, 524
registering definition, 498
setup, 498, 499
tags, 524
xScale, 604

W

W3schools, 405
Waterfall chart, 444
WDeploy, 107
First-hand knowledge.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.

Dwain Chang is an SAP BI consultant, originally from Surinam, who currently lives in the Netherlands. He has a master’s degree in informatics and economics as well as in marketing from the Erasmus University Rotterdam. Dwain started his professional career in 2011 and participated in various SAP backend and frontend development projects.

Xavier Hacking is an SAP BI specialist from Eindhoven, the Netherlands, and works as a consultant for Interdobs. He has a master’s degree in Industrial Engineering and Management Science from the Eindhoven University of Technology. He has worked with a wide range of products from the current SAP BW and SAP BusinessObjects BI toolsets, with a focus on dashboard development within SAP environments.

Jeroen van der A is an SAP BI consultant from the Netherlands working for Interdobs. He has more than 15 years of experience with business intelligence and started using SAP products in 2005. Jeroen is focused on finding innovative ways to use BI products to create added value, and uses a broad range of products to achieve this goal.