Unit 4

Creating Stories using Analytic Application in SAP SAC

Creating Analytic Applications in SAP Data Warehouse Cloud

Create an Analytic Application based on an SAP Data Warehouse Space. You can choose an SAP Data Warehouse space and navigate to SAP Analytics Cloud to create the application by selecting a dataset that is connected to the space and the space connection for a data-based widget such as table and chart.

Create Analytic Applications in SAP Data Warehouse

As a user of both SAP Analytics Cloud and SAP Data Warehouse, you can create an analytic application starting in SAP Data Warehouse, either based on SAP Analytics Cloud space or SAP Data Warehouse space.

Integration to SAP Data Warehouse

There are several scenarios to work with SAP Analytics Cloud and SAP Data Warehouse. For further information, please see chapter *Integration to SAP Analytics Cloud* in the *Integration Guide* of SAP Data Warehouse.

Please assure that the following conditions are made:

- To use SAP Analytics Cloud within the context of an SAP Data Warehouse space (Space Aware Connectivity), SAP Data Warehouse and SAP Analytics Cloud need to be on the same tenant.

- To use an SAP Data Warehouse analytical dataset, at least one live connection to an SAP Data Warehouse system needs to be configured in SAP Analytics Cloud.

Procedure

1. Starting in SAP Data Warehouse, select the **Files** start page.

2. Under **Spaces** select an SAP Data Warehouse space.

3. In the menu bar select + **Create** **Analytic Application**.

4. An upcoming dialog informs you that the application can't be created in SAP Data Warehouse and asks you whether you want to create it in SAP Analytics Cloud. Select **Create**.

   The analytics designer of SAP Analytics Cloud is displayed where you can start to create an application. In the URL header you can also see the space ID of the SAP Data Warehouse space you've selected in step 2.

5. Insert a model-based widget like a table or chart.

   A dialog is displayed where you can select an analytical dataset that is connected to the space and the space connection.

6. Select the dataset you would like to work with.

7. Choose **OK**.

8. Create the analytic application according to your needs.

9. Save the analytic application.

10. Select **Run Analytic Application** in the upper right corner.

    The application opens in another web browser window.

Creating an Analytic Application based on an SAP Analytics Cloud Space

You can choose the SAP Analytics Cloud space and navigate to SAP Analytics Cloud, where you can choose whether the table or chart in your application is based on an existing data model or on an analytical dataset. For the dataset option you're asked to select a configured connection, space and finally the dataset itself.

Procedure

1. Starting in SAP Data Warehouse, select the **Files** start page.

2. Under **Spaces** select the **Analytics Cloud** space.

3. In the menu bar select **Create** **Analytic Application**.

4. An upcoming dialog informs you that the application can't be created in SAP Data Warehouse and asks you whether you want to create it in SAP Analytics Cloud. Select **Create**.

   The analytics designer of SAP Analytics Cloud is displayed where you can start to create an application. If an application is created in the SAP Analytics Cloud space, the space ID won't be shown in the URL.

5. Insert a model-based widget like table or chart.

   A dialog is displayed where you have several options to select a model.

6. In the **Select Model** dialog, select one of the options:

   - **Existing Model**
   - **No Model** (only displayed if you have selected a table widget)

- **SAP Data Warehouse Analytical Dataset**

  If you decide to create your application from an analytical dataset, you first have to select a connection and a space.

7. Choose **OK**.

8. Create the analytic application according to your needs.

9. Save the analytic application.

10. Select **Run Analytic Application** in the upper right corner.

    The application opens in another web browser window.

Defining Busy Indicator

The sap. m. Busy Indicator busy indicator blocks specific UI areas that are defined by a control. For example, if a table in a complex UI is loading, only the table is blocked - the user can still carry on working with the rest of the UI.

You use busy indicators to inform users that something is going on in the background, for example, some data is being fetched from the back end and the user has to wait. As long as the busy indicator is shown, either all or a specific part of the UI is blocked, and no user interaction is possible.

Whenever busy indication is triggered, the default delay until the busy indicator is displayed on the UI is 1000 ms (1 second). If this delay were not in place, the busy indicator would always be displayed, even if there is no negotiable waiting time.

You can choose between the following busy indicators, depending on your use case:

- sap.ui.core.BusyIndicator
- sap.m.BusyDialog
- sap.m.BusyIndicator

Blocking the Whole UI

You can use the sap.ui.core.BusyIndicator busy indicator to block the whole UI. You can set the delay in ms by specifying the number:

```
sap.ui.core.BusyIndicator.show(<number>);
```

To release the UI again, the busy indication must be hidden again. This function call hides the busy indication immediately:

```
sap.ui.core.BusyIndicator.hide();
```

Busy Indication with Dialog

With the sap.m.BusyDialog busy indicator, you can block the whole UI like you do with sap.ui.core.BusyIndicator, but you can also show a dialog box. In this dialog box, you

can also include a Cancel button that users can choose to stop the activity that's running in the background.

If a control is set to busy, the complete control will be covered with a block layer, so no mouse events or keyboard interaction with the control are possible. If keyboard navigation is being used to step through the controls, controls that are set to busy are skipped and the focus jumps to the next control.

Here's how to do it:

```
var oMyListBox = new sap.ui.commons.ListBox({
   tooltip : "Country",
   editable : false,
   width : "200px",
   height : "200px",
   items : [ new sap.ui.core.ListItem({
      text : "I'm an item, and you?"
   }) ]
}).placeAt("uiArea1");

oMyListBox.setBusy(true);
```

The following code shows how you define the default state of a control as busy so that it will be displayed as busy when it has been rendered:

```
var oMyListBox = new sap.ui.commons.ListBox({
   busy : true,

   tooltip : "Country",
   editable : false,
   width : "200px",
   height : "200px",
   items : [ new sap.ui.core.ListItem({
      text : "I'm an item, and you?"
   }) ]
}).placeAt("uiArea1");
```

To release the control's busy state again, the same API can be used. This has to be done by the application after some data has been loaded, for example with the following command:

```
oMyListBox.setBusy(false);
```

To change the default delay of the local busy indicator, use:

```
oMyListBox.setBusyIndicatorDelay(<number>);
```

You can find some samples in the **Explored** app in the Demo Kit under sap.ui.core.Control.

Using Popups

Create Popups or Dialogs (Optimized Story Experience)

As a story developer, you can use popups and dialogs to design interactive stories.

Context

A popup helps viewers quickly enter information, perform configurations or make selections. You can also use it to display more specific data for a selected item on a story page. A popup acts as a container, so you can put any other widgets into it, such as table, button or checkbox.

You can also turn the popup into a dialog, which has the look and feel consistent with other dialogs in SAP Analytics Cloud.

Note

Currently, popups aren't available to story designers.

Procedure

1. Open the **Outline** panel. From **View** in the toolbar, make sure that **Left Side Panel** is selected.

2. In the **Outline** panel, hover over the page from which you want viewers to open the popup.

3. Select **(More)** **Add Popup**.

   You can see that an empty popup opens with the default name **Popup_1**.

4. If you want to use dialog for the popup, in the **Builder** panel of the popup, select **Enable header & footer**. Otherwise, skip to step 7.

5. **Optional:** You can change the title and configure buttons for the dialog.

6. Choose **Apply**.

7. Add widgets to the popup, such as dropdown boxes, tables or charts.

   For dialogs, you can only add widgets to areas other than the header and footer.

   Note

   If there're no widgets in the popup, even if you save the story with it, viewers won't see it even if they've triggered it.

8. Edit the styling of the popup in the **Styling** panel. For example, you can change its height and width.

Results

Furthermore, in the **Outline** panel, you can select fx to the right of the popup or a widget in the popup to add scripts to it. For example, they can trigger events when viewers click a button in the popup.

To return to a page, select it from either the page tabs or **Outline**. You can also reopen the popup by selecting it from **Outline**.

To delete the popup, find it in the **Outline** panel, and choose ••• **(More)** 🗑 **Delete**.

Scripting in SAP SAC Analytic Applications

Most modern analytics tools avoid scripting to simplify the designer's tasks. Users may find it easier to use at first, but they quickly find themselves limited to the scenarios built into the tool.

Scripting allows you to go beyond present narratives, to respond to user interaction in a custom way, to change data result sets, and to dynamically alter layout. Scripting frees your creativity.

The scripting language in Analytics Designer is a limited subset of JavaScript. It is extended with a logical type system at design time enforcing type safety. Being a true JavaScript subset allows executing it in browser natively. All scripts are run and validated against strict mode. Some more advanced JavaScript features are disabled. Scripts are either tied to events or global script objects.

Type System

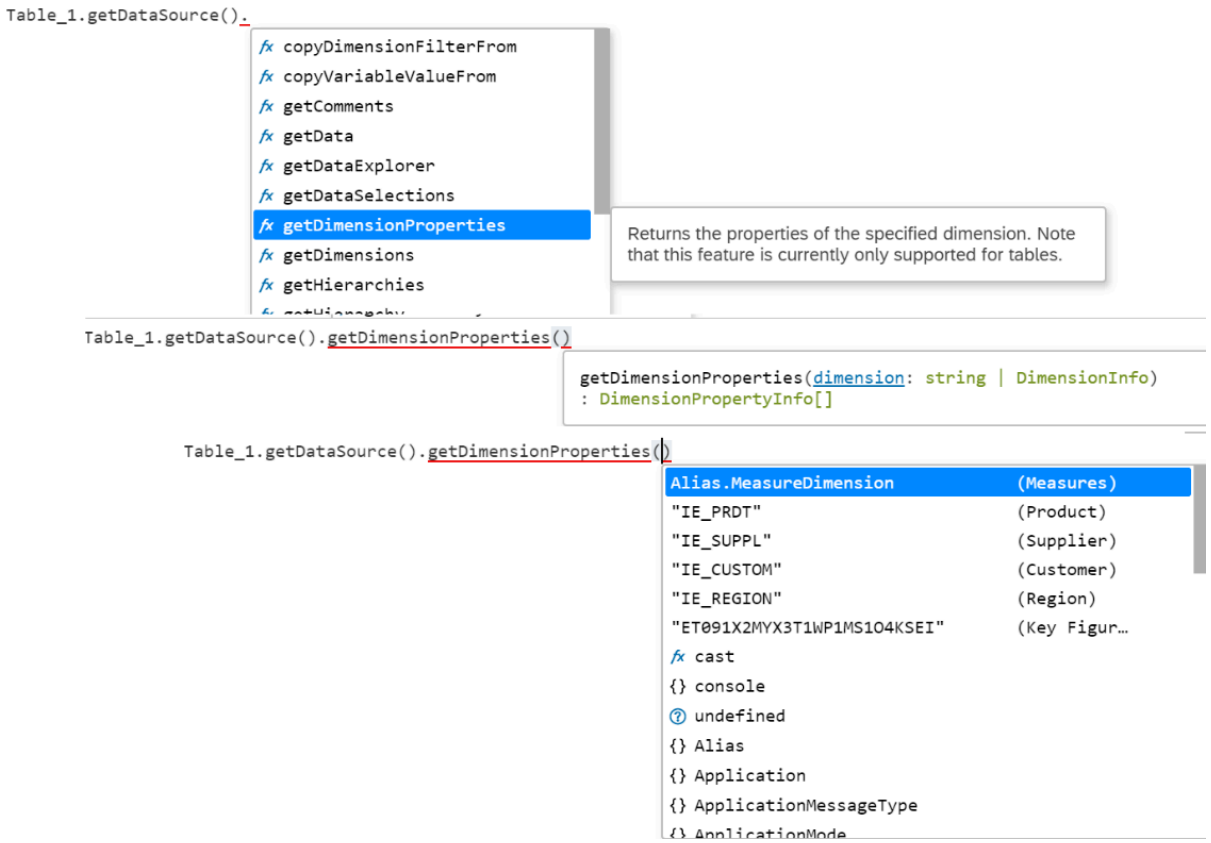
The logical type system runs on top of plain JavaScript. It enforces strict types to enable more powerful tooling. The behavior at runtime doesn't change as it is still plain JavaScript.

Tooling – Code Completion and Value Help

The Analytics Designer scripting framework exposes analytics data and metadata during script creation and editing. This enables:

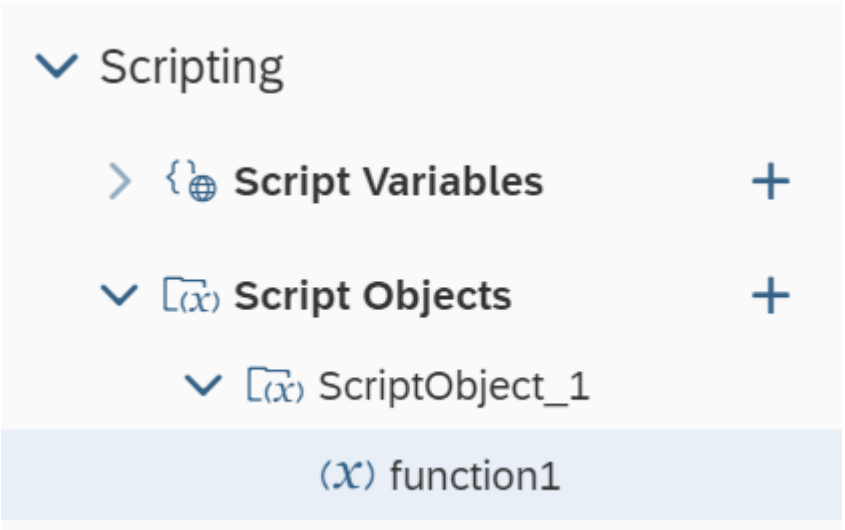
- Code completion in the traditional sense like completing local or global Identifiers
- Semantic code completion by suggesting member functions or similar
- Value help in the form of context-aware value proposals like measures of a data source for function parameters

For example, when calling an API method on a Business Warehouse DataSource, the code completion can propose measures as code completion options or values to specify a filter.



Understand the Global Script Objects & Script Variables

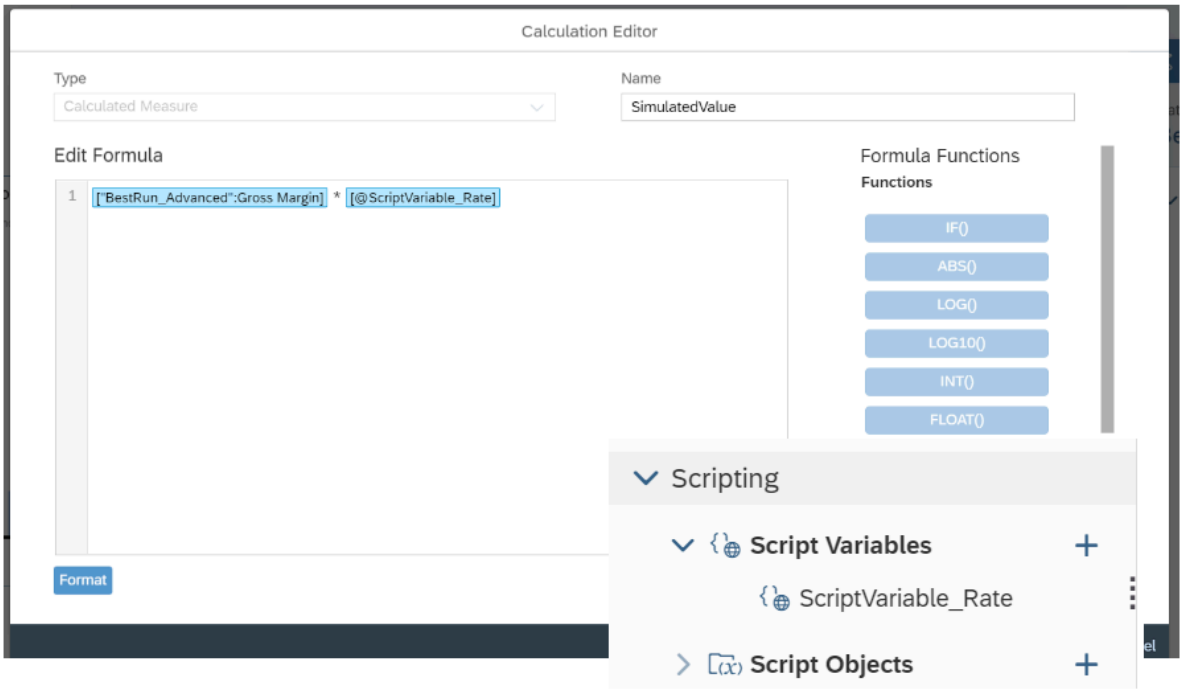
Global script objects act as containers. They allow you to maintain and organize script functions that are not tied to any event and are invoked directly. You can maintain libraries of re-usable functions. These library scripts are called functions.



You can access every object in the Outline pane such as widgets, script variables, or script objects by its name when you are working on a script.

By referencing Script Variable in Calculated Measure, users can easily build a what-if simulation with query results.

For example, an analytic application developer can bind a calculated measure which references one script variable (ScriptVariable_Rate) to a chart.



The Core Scripting Language Features

Dynamic typing means that the runtime will try to guess the type from the context at that moment and the user can even change the type after the variable is already in use. For example, you could change the value of the before-mentioned integer to another type of object at will; “Dear integer, you are now a duck”.

SAP Analytics Cloud, analytics designer forbids both. Once you have a duck, it remains a duck and you can’t recycle variable names as new types. If you want something else, you’ll need another variable. It is also strongly typed, meaning that if you want to use an integer as a string, you’ll have to explicitly cast it. Both are a consequence of enabling the rich code completion capabilities in the editing environment.

The analytics designer scripting language is still JavaScript. You can write perfectly valid JavaScript while treating the language as if it was strongly and statically typed.

No Automatic Type Casting

A consequence of strong typing is that you can’t expect automatic conversions. The following is valid JavaScript:

```
JavaScript

Copy

var nth = 1;
console.log("Hello World, " + nth);
```

In analytics designer, you will see an error in the script editor, informing you that auto-type conversion is not possible, and the script will be disabled at runtime, until fixed. Instead, you should explicitly cast nth to a string.

```
JavaScript

Copy

var nth = 1;
console.log("Hello World, " + nth.toString());
```

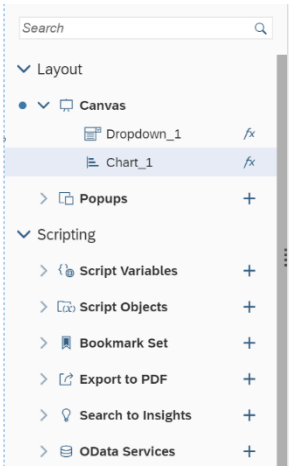
Accessing Objects

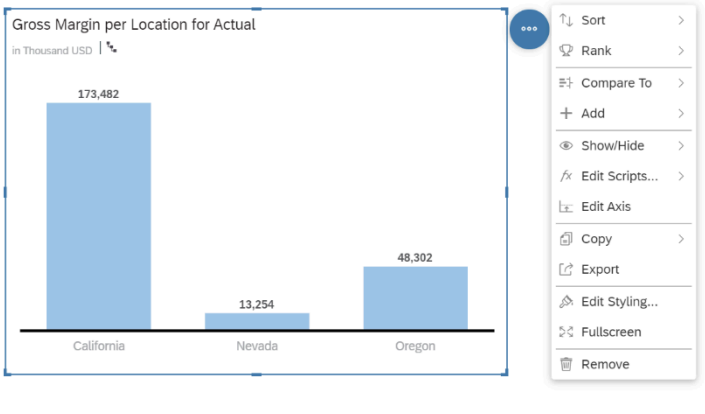
Every object (widget or global script object) is a global object with the same name as in the outline. Suppose you have a chart in your application, named Chart_1 and want to check and see if it is visible. You can access Chart_1 as a global variable and then access its functions, in this case to see if it is currently visible.

```
JavaScript

Copy

var isVis = Chart_1.isVisible();
```





Finding Widgets with Fuzzy Matching

The application author can type in the complete name of a widget or just some first letters. By typing CTRL+Space, the system either:

- Completes the code automatically if there is only one valid option
- Displays a value help list from which you can select an option

Fuzzy matching helps you finding the result even if you have made a typo or the written letters are in the middle of the function. Fuzzy matching is applied automatically for the standard code completion (for example, “cose” → “console”).The script validation runs automatically in the background and shows errors and warnings indicated with red and orange underlying and a red or orange marker before the line number.

External Libraries

There is no provision in SAP Analytics Cloud, analytics designer for importing external JavaScript libraries. You can use the standard JavaScript built-in objects such as:

- Math
- Date
- Number
- Array
- Functions on String

All standard functions listed in the SAP Analytics Cloud, analytics designer API Reference are supported even if some browsers don’t support them natively.

For example, *String#startsWith* is not available in Microsoft Internet Explorer, but can be used in SAP Analytics Cloud with all browsers.

Two types of JavaScript loops are possible in SAP Analytics Cloud, analytics designer, *for* and *while* loops. Other types, such as foreach iterators, are not supported.

For loops

For loops are standard JavaScript *for* loops, with one caveat. You must explicitly declare the for iterator. This is valid JavaScript, but it isn’t accepted in the script editor:

JavaScript

Copy

```
for (i = 0; i < 3; i++) {
console.log("Hello");
```

```
}
```

Instead, explicitly declare i with the var keyword. The example below is valid:

```
JavaScript
Copy
for (var i = 0; i < 3; i++) {
console.log("Hello");
}
```

While loop

SAP Analytics Cloud Analytics Designer fully supports *while* loops:

```
JavaScript
Copy
var nth = 1;
while (nth < 3) {
console.log("Hello while, " + nth.toString());
nth++;
}
```

For in loop

An additional type of loop is the *for in* iterator. Suppose you had a JavaScript object: you can iterate over the properties with the for in loop. Data selections are JavaScript objects and can be iterated over:

```
JavaScript
Copy
var selection = {
"Color": "red",
"Location": "GER"
};
for (var propKey in selection) {
var propValue = selection[propKey];
...
```

```
};
```

Create a Script Object and Script Function

Procedure

1. In the **Scripting** area of the **Outline** panel, (for analytic applications) choose + right next to **Script Objects**, or (for optimized stories) choose + **Script Objects**.

   The new script object is displayed under **Script Objects**, and a **Script Function** panel opens.

2. In the **Properties** section, configure the following settings for the script function:

Element	Description
Name	The name is mandatory and it validates according to the same rules as for global variables. Name can't be empty or a reserved keyword and should begin with a letter.
Description	(optional) A brief description of the function.
Return Type	Can be **void**, **string**, **boolean**, **integer** or **number**. By default, the return type is **void**. Besides the primitive types described above, you can also select a wide variety of non-primitive types like **Button**, **Category**, **Chart**, **Clock**, **DataSource**, **Table**, **UrlType**.
Set as Array	Only activated if the return type is not void.

3. **Note**

4. All modifications take effect immediately without your having to select **Done**, for example, after you press enter on an input, change the dropdown selection or switch to an array.

5. In the **Arguments** section select ⋯ to create a new argument for the function.

6. In the **Argument** panel, edit name and type of the argument.

   If the argument should represent a list of this type, set it to array.

7. Choose **Done** to close the **Argument** panel and return to **Script Function**.

8. **Optional:** You can reorder the arguments by dragging and dropping them inside the panel.

   This changes the order of the arguments in the defined function.

9. In the **Outline** panel select fx (**Edit Scripts**) right next to the script object to write scripts for the new script function.

Results
You've created a script object. You can use it in any widget event handler.

You can add multiple script functions to the script object by selecting ••• + **Add Script Function** next to it in the **Outline** panel.

Example
The following example displays the elements of the script function computeAverage.

- name: computeAverage
- return type: **number.**
- arguments: value1, value2 and value3 (all **number** type)
- scripts: return (value1 + value2 + value3) / 3.0;

Modify And Delete Script Objects and Script Functions

To change the name, description, arguments and return type of a script function, select it in the **Scripting** area of the **Outline** panel. The panel opens where you can directly update your changes. After you've changed the elements, select **Done** to close the panel.

To edit an argument, in the **Arguments** section of the **Script Function** panel, select (**Edit**) when hovering over it. To remove an argument, select ⊗ (**Remove**) next to it.

To delete a script object or script function, select ••• next to it and then **Delete**.

Write Scripts in Script Editor

Script editor lets you, as an application designer or story developer, write scripts for each widget and thus create interactive and highly custom-defined analytic applications or optimized stories.

Context
To enable interactivity, you configure the behavior of widgets and write scripts that are executed when the viewer performs an action in the analytic application or story. For

example, you can add a button widget and assign scripts to the its onClick event. You can also write scripts based on other system events like the onInitialization event of the application or page or scripts that are executed whenever data is changed.

Scripts consist of one or more statements written in a JavaScript-based language, which follow a specific syntax. You write scripts in the script editor. You can find all objects, fields and functions available for scripting in Analytics Designer API Reference (for analytic applications) or Optimized Story Experience API Reference (for optimized stories).

Procedure

1. Add a widget to the canvas or page, for example, a dropdown.

   The widget icon and name are displayed in the Outline panel.

2. Hover over the widget name.

   An fx button is displayed right next to the widget name.

3. Select fx. If the widget supports multiple events, select fx and then one of the events that triggers the execution of scripts.

   You can also select fx in the widget's ••• (More Actions) menu.

   The script editor opens in a new tab, which displays the name of the event and the widget, page or application to which the script will be assigned, for example, Table_1 - onSelect.

   You can change the order of the multiple tabs by dragging and dropping them horizontally.

4. Type in one or more statements with this syntax: <ComponentVariable>.<function>(<arguments>);.

   Tip

   You can call the value help at any place in the script by pressing CTRL + Space .
   Sample Code

```
//Gets the selected text of a dropdown.
var selectedText = Dropdown_1.getSelectedText();
```

When you've finished your script, close the tab by selecting ✕ at its right side. **What is the syntax for creating a new argument for a function in SAP SAC?**

To create a new argument for a function in SAP SAC, you need to add a parameter in the function definition. The syntax for adding a parameter is as follows:

functionName(parameterName: parameterType).

Add multiple arguments to a function in SAP SAC

It adds multiple arguments to a function in SAP SAC. In fact, it is common to have functions that require more than one argument to perform their intended task. When defining a function in SAP SAC, you can specify the number and type of arguments that the function should accept. These arguments are then passed to the function when it is called, allowing it to perform its task using the provided data. To create a new argument for a function in SAP SAC, you can simply add a new parameter to the function definition and specify its type and name.

Specify the data type for a new argument in SAP SAC

To specify the data type for a new argument in SAP SAC, you can follow these steps:

1. Open the function you want to add a new argument to.
2. Click on the "Edit" button to enter the edit mode.
3. Click on the "Add Argument" button to add a new argument.
4. In the "Argument Name" field, enter the name of the new argument.
5. In the "Data Type" field, select the data type you want to assign to the new argument. The available data types include string, integer, decimal, boolean, and date.
6. Click on the "Save" button to save your changes.

By following these steps, you can specify the data type for a new argument in SAP SAC.

It possible to define default values for function arguments in SAP SAC?

Yes, it is possible to define default values for function arguments in SAP SAC.

When defining a function in SAP SAC, you can specify default values for the function parameters. This means that if a user does not provide a value for a particular parameter when calling the function, the default value will be used instead.

To define a default value for a function parameter in SAP SAC, you can simply assign a value to the parameter when defining the function. For example, if you have a function with a parameter called "myParam", you can set a default value for it like this:

javascriptCopy

```
function myFunction(myParam = "default value") {
  // function code here
}
```

In this example, if the user does not provide a value for "myParam" when calling the function, the default value of "default value" will be used.

Pass values to the arguments of a function in SAP SAC?

To pass values to the arguments of a function in SAP SAC, you can follow these steps:

1. Define the function with its arguments in the script editor.
2. Call the function and pass the values to its arguments when invoking it.

Here's an example code snippet to illustrate this:

javascriptCopy

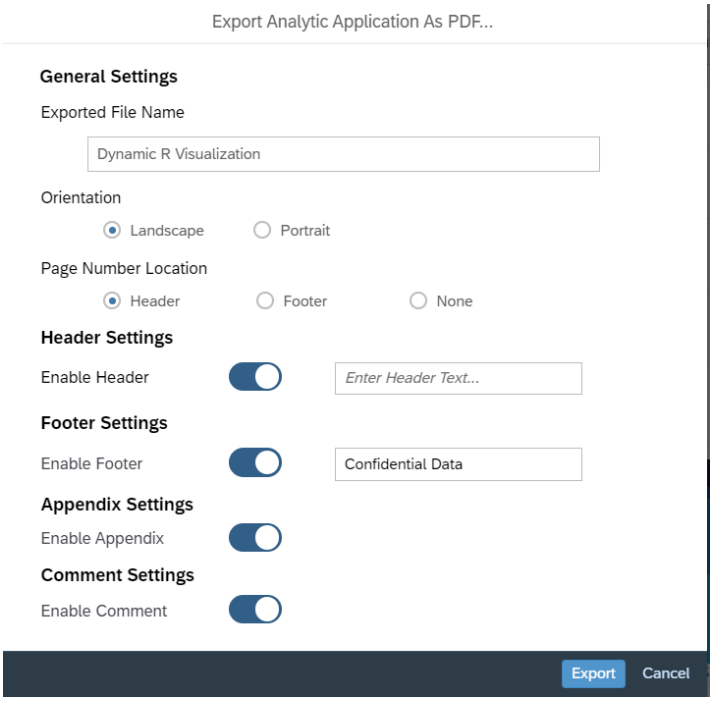
```
// Define the function with its arguments
function myFunction(arg1, arg2) {
  // Function code here
}

// Call the function and pass values to its arguments
var value1 = "Hello";
var value2 = "World";
myFunction(value1, value2);
```

In this example, myFunction is defined with two arguments arg1 and arg2. When calling the function, we pass the values "Hello" and "World" to these arguments respectively.

Save and Run the Analytic Application

Application users can export whatever appears on the application running page via clicking the image created in Step 9 and select the required properties, click Export.



Please note that only visible contents in your application can be exported to PDF. The below invisible elements won't be exported:

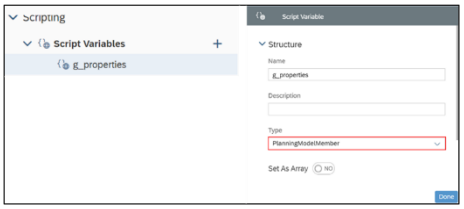
- Invisible part in scrollable charts, tables or tab strips

- Collapsed table cells
- Lazy rendered widgets
- Comments on invisible data cells
- Custom widgets and web page widgets

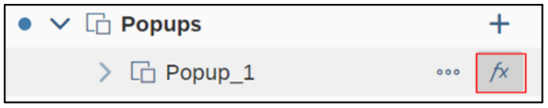
Scripting in SAP SAC Analytic Applications

Create Script in Analytic Application
I want the Target Steps, that are entered in the Input Field to be saved, when the user clicks on 'OK' in the Popup.

Before adding the script, I also created a variable, which is called g_properties and which is of type 'PlanningModelMember'.



Now, I add the coding as OnClick for the Popup.




```
//only execute script if the user clicks on OK
if ( buttonId === "button1" ) {

//save new target steps to Dimension User Properties
var target_steps = Input_NewTarSteps.getValue();
g_properties=({id:Var_UserID,properties:{TargetSteps:target_steps}});
PlanningModel_1.updateMembers("User",g_properties);
Popup_1.close();};
```

If the user enters Target Steps in the Input Field and presses the OK button, the Dimension 'User' will be updated in the background.

Widgets in SAP SAC overview

A custom widgets are implemented as Web Components. The Web Components are made of HTML, CSS and JavaScript.

Custom Widgets

Custom widget consists of the following files:

- **Custom Widget JSON**
  The custom widget JSON file specifies the custom widget properties like id, version, name, description and so on.
- **Web Component JavaScript**
  A custom widget is composed of one or more Web Components.
  Each Web Component is implemented in a Web Component JavaScript file, which defines and registers a custom element and implements the custom element's

JavaScript API. It has the lifecyles: constructor(), onCustomWidgetBeforeUpdate(), onCustomWidgetAfterUpdate(), connectedCallback().

- **Web Component JavaScript of Styling Panel (optional)**
  The Styling Panel of a custom widget is an area in analytics designer where you can set property values of the custom widget at design time. It is implemented as a Web Component.
- **Web Component JavaScript of Builder Panel (optional)**
  The Builder Panel of a custom widget is an area in analytics designer where you can set property values of the custom widget at design time. It is implemented as a Web Component.
- **Icon file**
  Any icon file image in 16×16 pixels.

Prerequisites

You need a web server that hosts the resources of the custom widget files (JavaScript files, CSS files, images, and so on). Assume that your web server address is :

```
https://example.demo/customwidgets
```

Create a Custom Widget

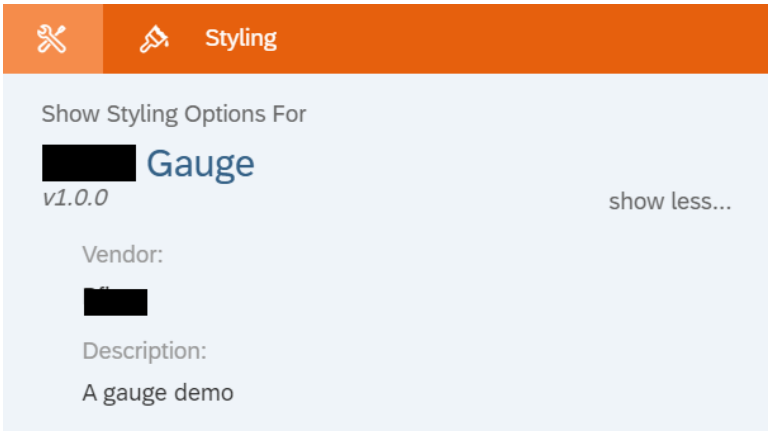
The Gauge Box custom widget consist of three Web Components: the actual Gauge Box, the Styling Panel and the Builder Panel of the Gauge Box and it consist the following files:

- **box.json**
  Custom Widget JSON of Gauge Box.
- **box.js**
  Web Component JavaScript file of the Gauge Box.
- **box_sps.js**
  Web Component JavaScript file of the Styling Panel of the Gauge Box.
- **box_bps.js**
  Web Component JavaScript file of the Builder Panel of the Gauge Box.
- **icon.png**
  Icon of the Gauge Box in any 16×16 pixel icon.

1. Custom Widget JSON of Gauge Box (box.json)

The Gauge Box custom widget has the unique ID, version, and the name, which is displayed in the analytics designer, Styling Panel.

```
"id": "com.demo.gauge",
"version": "1.0.0",
"name": "Demo Gauge",
"description": "A gauge demo",
"newInstancePrefix": "Gauge",
"icon": "https://example.demo/customwidgets/box/icon.png",
"vendor": "Demo",
"eula": "EULA",
"license": "1.0",
```



The Gauge Box custom widget is composed of the following three Web Components:

```
"webcomponents": [
    {
        "kind": "main",
        "tag": "com-demo-gauge",
        "url": "https://example.demo/customwidgets/box/box.js",
        "integrity": "",
        "ignoreIntegrity": true
    },
    {
        "kind": "styling",
        "tag": "com-demo-gauge-sps",
        "url": "https://example.demo/customwidgets/box/box_sps.js",
        "integrity": "",
        "ignoreIntegrity": true
    },
    {
        "kind": "builder",
        "tag": "com-demo-box-bps",
        "url": "https://example.demo/customwidgets/box/box_bps.js",
        "integrity": "",
        "ignoreIntegrity": true
    }
],
```

The first Web Component is the actual Gauge Box as indicated by the kind of “main”. The second Web Component is the Styling Panel of the Gauge Box as indicated by the kind of “styling”. The third Web Component is the Builder Panel of the Gauge Box as indicated by the kind of “builder”.

Moving on, these are the properties of the Gauge Box custom widget: **value**, **info**, **color**, **width** and **height**.

```
"properties": {
    "value": {
        "type": "number",
        "description": "Gauge value",
        "default": "0"
    },
    "info": {
        "type": "string",
        "description": "Gauge info",
        "default": ""
    },
    "color": {
        "type": "string",
        "description": "Text Color info",
        "default": "#3498db"
    },
    "width": {
        "type": "integer",
        "default": 100
    },
    "height": {
        "type": "integer",
        "default": 100
    }
},
```

The property value represents the value in percentage of the Gauge Box. The property info represents the title of the Gauge Box. The property color represents the color of the Gauge Box. And the properties width and height represent the initial width and height of the custom widget.

And then the script methods of the Gauge Box are defined:

```
"methods": {
    "setValue": {
        "description": "Sets the Gauge value.",
        "parameters": [
            {
                "name": "newValue",
                "type": "number",
                "description": "Gauge value"
            },
            {
                "name": "newInfo",
                "type": "string",
                "description": "Gauge info"
            },
            {
                "name": "newColor",
                "type": "string",
                "description": "Text Color info"
            }
        ],
        "body": "this.value = newValue; this.info = newInfo; this.color = newColor;"
    },
    "getValue": {
        "returnType": "number",
        "description": "Returns the Gauge value.",
        "body": "return this.value;"
    }
},
```

The function **setValue** takes three parameters, **newValue**, **newInfo** and **newColor**. The body property contains the script code which sets the passed all the parameters to the respective Gauge Box's properties.

Function **getValue** takes no parameter and returns the percentage value of the Gauge Box.

Finally, an **onClick** event is defined:

```
"events": {
    "onClick": {
        "description": "Called when the user clicks the Box."
    }
}
```

Note that the event has no parameters.

Uses of Widgets in SAP SAC

Use Custom Widgets in Analytic Applications

You can use custom widgets, which extend the functionalities of SAP Analytics Cloud, analytics designer and complement the standard palette of widgets according to your needs.

Prerequisites
To create and upload custom widgets, the **Create** permission for **Custom Widget** must be selected in the role that you are assigned.

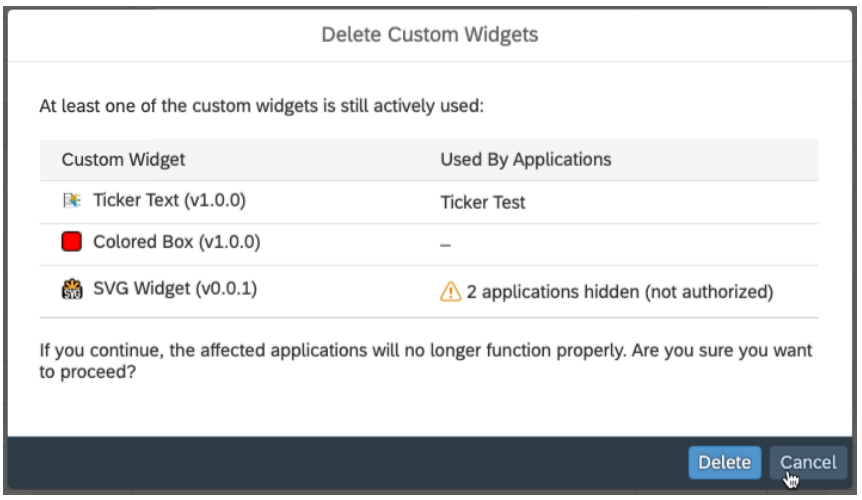
Procedure

1. On the **Analytic Applications** start page, choose the **Custom Widgets** tab.
2. Select + (**Create**).
3. In the **Upload File** dialog, choose **Select File**.
4. Select the custom widget file, for example box.json.

Results
When you want to insert the custom widget into your analytic application, you can find it via + **(Add) Custom Widgets**.

Next Steps
Every custom widget has a version number in the format majorVersion.minorVersion.patchVersion defined in its original file, for example, 1.0.0. You can add different major versions of a custom widget to analytics designer at the same time, for example, versions 1.0.0 and 2.0.0.

However, when you add a custom widget that differs in minor version from the one present in analytics designer, it replaces the present one. For example, if a custom widget of version 1.5.0 is present in analytics designer, then adding either version 1.4.0 or 1.6.0 replaces version 1.5.0.



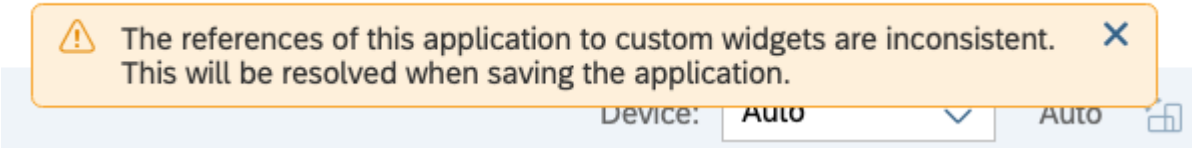
Example

In the following example, the first custom widget is used by an application named **Ticker Test**, the second one isn't used in other applications, and the third one is used in two applications for which the you have no authorization.

Note

If you choose to ignore the warning and delete the custom widget regardless of active use and later realize that this was a mistake and re-upload the custom widget, the references between applications and this custom widget will stay lost.

This means that the export of the custom widget with the analytic application actively using it and the display of related applications in the dialog won't work anymore. In this case, you'll see a warning when opening an affected application:



To solve this, just save the application. You can therefore repair the dependencies and will no longer see the warning.

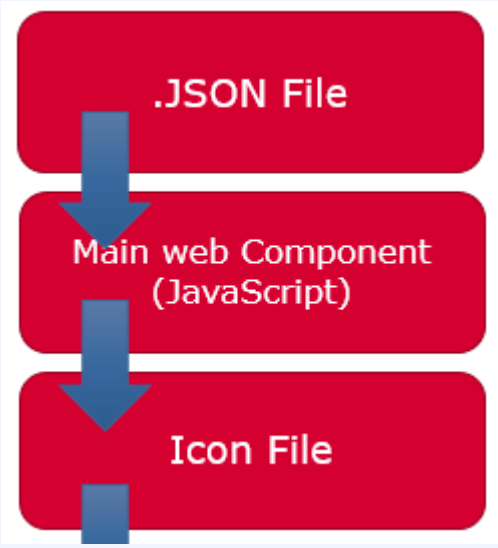
Create a Custom Widget, Modifying the Custom Widget

What goes into creating a Custom Widget?

At their simplest, custom widgets are a collection of different files which, when read together, tell SAC what additional visualisation and functionality is available to the user and how to execute this functionality within a story or analytical application.

Some files are mandatory, and others are only required depending on the type and scope of functionality that you want your widget to offer.

The simplest example of a custom widget just requires a .JSON File, a Main Web Component (JavaScript File) and an Icon File:



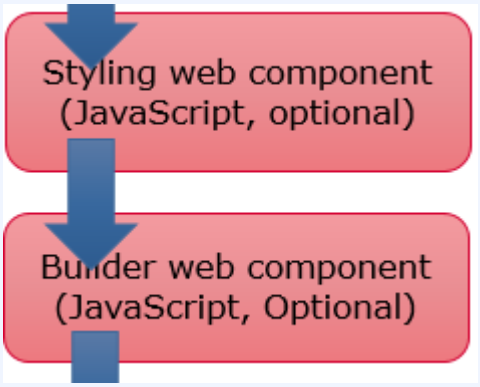
The .JSON file contains the metadata of your custom widget in the form of tags, Id's, methods, events and other properties. It also tells SAC where to find the other webcomponent files (Main, Styling, Builder) used by the widget.

The Main webcomponent file is essential as this file contains the JavaScript that provides the core functionality to the widget by providing the code containing the relevant methods and events for you to use within SAC.

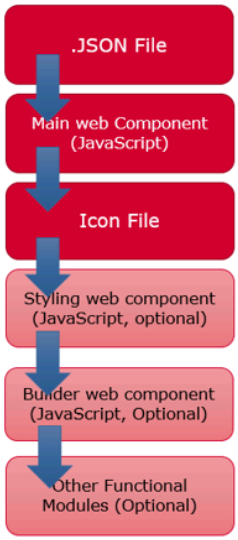
Finally, we have an Icon file which is used to provide an image to represent your custom widget within SAC.

Depending on your needs this could be sufficient, but to create a more complex widget, you might want to include a Styling Webcomponent file and a Builder Webcomponent file. These files will better enhance your widget giving it more functionality and purpose to your analytical application:

You can have the optional Styling Webcomponent file which allows you to enhance your custom widget's look and feel by making use of the styling pane within SAC.



You can also add another JavaScript file for an optional Builder Webcomponent, which is used to set certain property values and control functional settings of the custom widget.

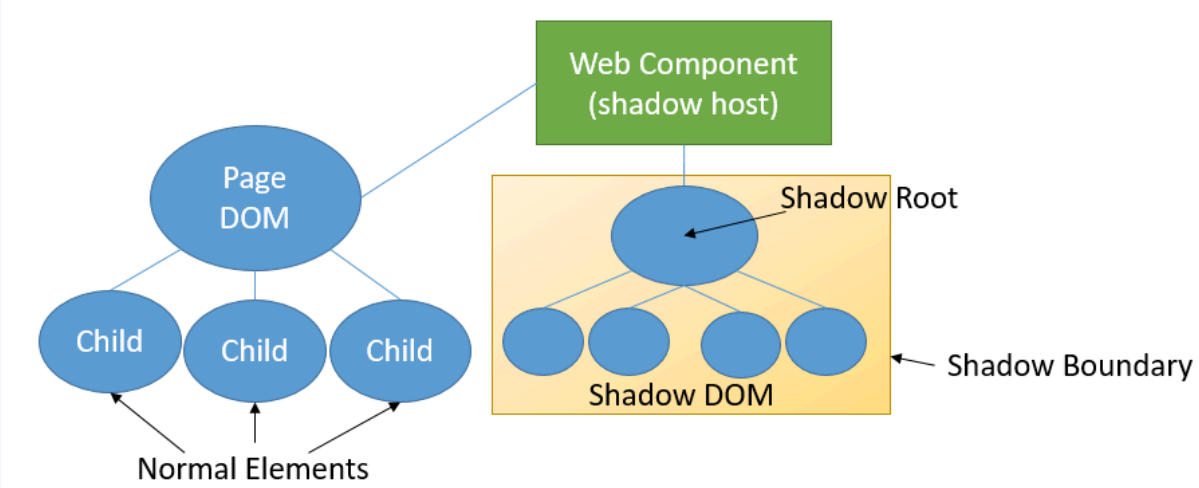


In addition, if you need to go even further in terms of functionality, then there is still the option to add more function modules to achieve your goals and better enhance your widget.

Incorporating custom widget code into SAC

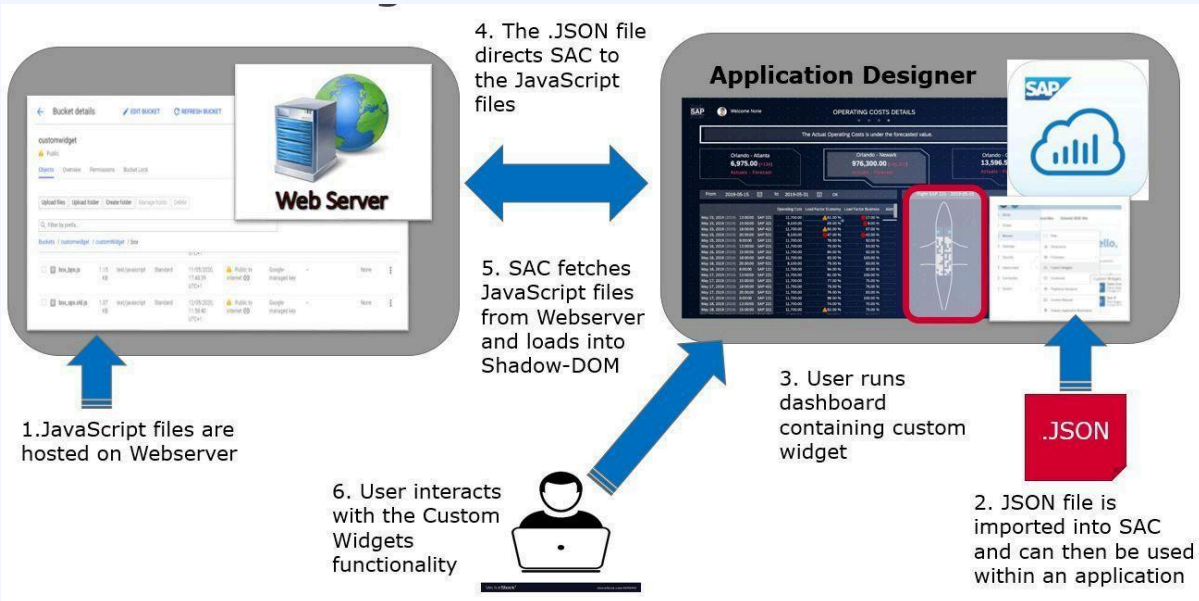
So how do you get your Custom Widget and its code into SAC? At the heart of the process is the use of a Shadow Document Object Model. This is a method by which you can encapsulate custom code within an application or an HTML page without affecting the main source code of the HTML page.

SAP has taken this route to ensure that you are able to change, add or remove from your widget without actually affecting the applications code. SAC does not allow you to interfere with the HTML & JavaScript of your SAC Tenant, so they allow you to use a shadow DOM where you are able to have added adjustments of the HTML code on the widget, but without affecting the SAC HTML and JavaScript.



This is the reason why we need to have access to an external webserver, as SAC doesn't allow us to host our files and code within SAC. We need somewhere to host our custom widget files so that SAC can access them when it needs to.

Working with custom widgets in SAC



1 All files apart from the .JSON file are stored on your external Webserver.

2 You upload your main .JSON file on to your SAC tenant and use it within a story or analytic application.
3 The end-user goes on to run the story or analytic application which contains the custom widget
4 This prompts SAC to use the custom widget's .JSON file and connect with the external webserver.
5 Your SAC tenant fetches the relevant files outlined in the .JSON file and loads them onto

the Shadow-Dom so the functional code and visualisations can be executed.
6 You are now able to interact with the widget within your analytic application.

SAC gives us a file format/structure you will need to follow to ensure that your widget works, it's ok, it's not as daunting as it sounds and it's just a simple process to follow to ensure that your widget is created.

How do I go about starting to develop my own?

Hopefully I've piqued your interest in custom widgets and explained why you might want to move away from the standard ones available to you and start thinking about what you could create yourself.

If you now want to create your own widget but don't know where to get started, there are tons of walkthroughs and blogs about custom widgets you can find, and I have highlighted some excellent blogs and walkthroughs which were beneficial to me.

Best Practices for SAP Analytics Cloud, Analytics Designer

SAP Analytics Cloud doesn't have to be a chore – here, you will find some best practices to keep Analytics Designer running smoothly. Some of these practices may be applicable universally in the application, while others are more catered towards specific situations. In any situation it is advised to test out these practices in a copy of your specific application prior to committing to any changes.

Analytics Designer and Stories both share the same widgets, such as tables, charts, and images. These practices will be useful for both portions of the application.

To get the most out of Analytics Designer, consider the following.

Application Design

Avoid one complex application containing many data sources.

- Build multiple applications and use application URLs to navigate from one application to the other using the Navigation Utils API.
- Try to divide one application into multiple tabs, panels, and so on, in order not to show each widget (and its data) of your analytic application at startup.
- If you have multiple numeric chart widgets that consume the same data model, consider enabling the Query Merge capability (In the toolbar, select File > Edit Analytic Application > Query Settings, then enable Enable Query Merge) or using the getData API to display key figure values. This improves performance by avoiding multiple requests to the backend.

Application Start-Up Mode

Start the analytic application in embed mode (by adding ;mode=embed to the analytic application's URL). This avoids loading and rendering the SAP Analytics Cloud shell. Expect a minor performance improvement here.

Avoid Duplicating Widgets Unnecessarily

- Consider using the Set Style API to change the font, color, background color, and so on, of widgets. Contrast the following patterns:

    - Bad pattern (due to limitations in former releases): To display a text in two different colors, for example, red and green, you create two text widgets, each with a different text color. Depending on the situation, you show the appropriate text widget and hide the other one.
    - Good pattern: You create a single text widget and apply the appropriate color with setStyle().
- Consider Moving Widgets

If your analytic application uses the same widget, for example, a table or chart, in several tab strips or panels, use the Move Widgets API to move this widget to the currently visible container, for example, a tab of a tab strip or a panel.

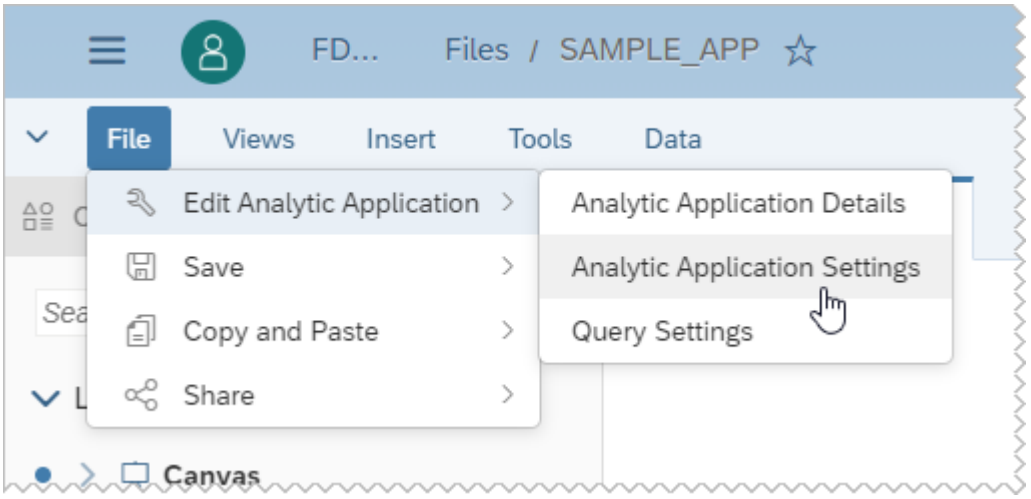
Load Invisible Widgets in Background

This provides a faster perceived startup of an analytic application containing (many) invisible widgets.

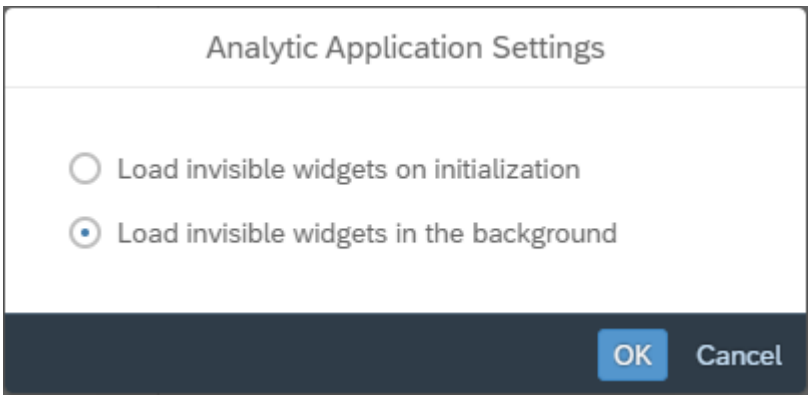
In former releases, all widgets (visible or invisible) had to be initialized before the analytic application started, that is, before the end user saw the startup screen.

As the application designer, you can change this default behavior by activating loading invisible widgets in the background. All widgets that are initially visible are initialized and displayed to the end user. After, the invisible widgets are initialized in the background to be available when the end user changes their visibility. Invisible widgets are not only widgets with the flag Show this item at view time turned off but also widgets inside invisible panels or on tabs in tab strips that are inactive at startup. This new loading behavior increases the perceived startup performance of the analytic application because the startup screen appears faster.

You can change to the new behavior by using the Analytic Application Settings dialog:



Then, select Load invisible widgets in the background:



You can overwrite this setting with the URL parameter loadInvisibleWidgets, This is useful, for example, if you as an application designer want to decide which mode is better working with your analytic application or if you as an end user aren't satisfied with the choice of the application designer. There are two possible values:

The following value forces the "classic" default behavior and loads all widgets before any of the widgets are displayed to the end user:

loadInvisibleWidgets=onInitialization

The following value forces the background loading of invisible widgets after initially the visible widgets are displayed to the end user:

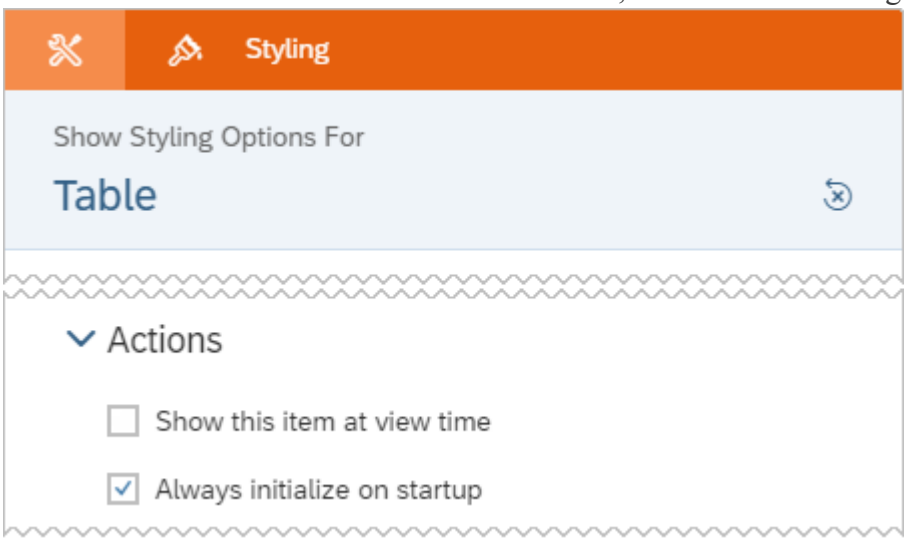
loadInvisibleWidgets=inBackground

Tips and Tricks for Loading Invisible Widgets in the Background

If the mode Load invisible widgets in background is used, it is possible that a script tries to access a widget which doesn't exist at this point in time. This applies especially to the onInitialization event script. Here are some best practices to reveal the full potential of this optimization:

- Leave the onInitialization event script empty.
- If this isn't possible, try to avoid accessing initially invisible widgets. Especially avoid using the setVariableValue() method if is not really necessary. If you need to set a variable initially to a static value, set the variable value state at design time in the analytic application or via a URL parameter instead.
- If you need to configure invisible widgets via scripting (that includes setting filters or variables), then do this directly before the widget becomes visible (for example, before calling setVisible(true) or before changing the tab in a tab strip).
- If you must access an invisible widget in the onInitialization event script, avoid nesting this widget too deep into a container structure, like, for example, in panels or tab strips. Ideally, the widget should be a direct child of the Canvas.
- If you must access an invisible widget in the onInitialization event script, select the Always initialize on startup checkbox of the widget. It is located in the Styling Panel of the widget.
  Note: If the widget is part of an invisible container, then select also the Always initialize on startup checkbox of the container.

Note: If an invisible panel has Always initialize on startup set to true, then it is initialized as if it was visible. In addition, its visible child widgets are also initialized.



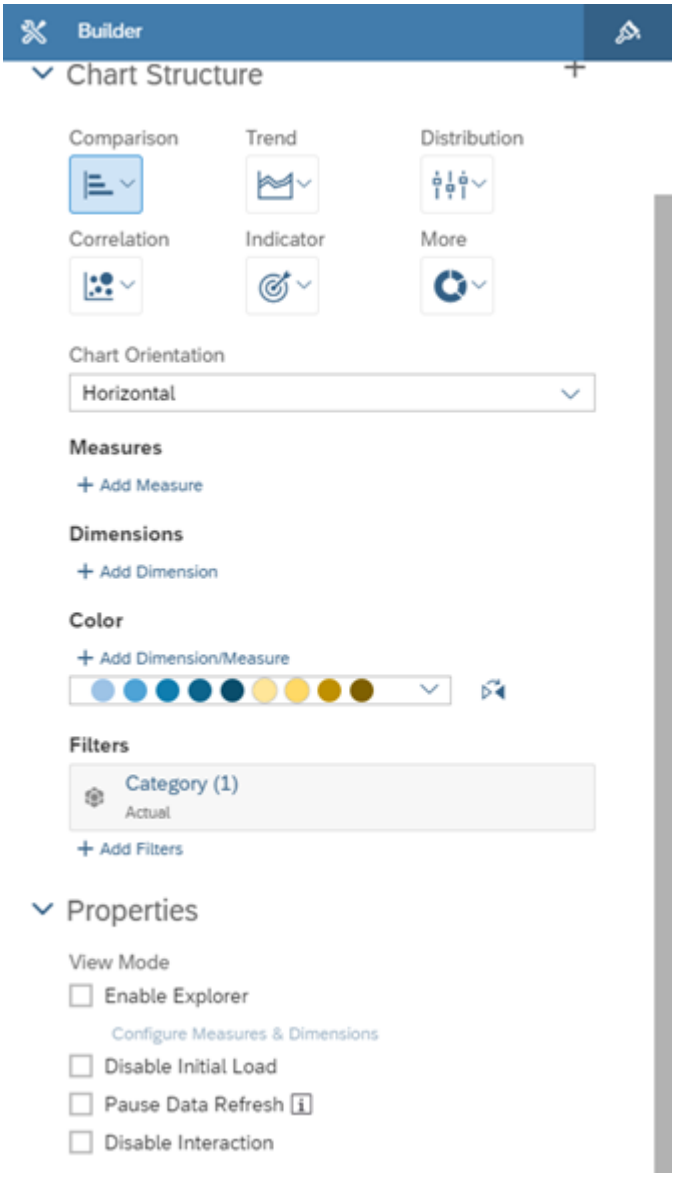
Use the Pause Refresh Script API to Optimize Application runtime Performance

Application designer or developer can build applications, allowing analytic application end users to pause the refresh of a Table or Chart widget until they have completed adding or removing dimensions, filters, and key figures to such a widget. This optimizes runtime performance because an unnecessary initial refresh is avoided, and remote queries won't be triggered every time when analytic application end users take an action.

Pause Refresh of a Chart and Table can be used in the following scenarios:

- Pause the initial refresh of applications until they are modified during application onInit event to optimize the application start up performance. Especially if setVariableValue action is defined in onInitialization event, you can use pause refresh API to pause the initial rendering of charts and tables and refresh them after setVariableValue action is completed.
- Pause the initial invisible widgets. Refresh the widgets after they start becoming visible. The initial rendering of invisible widgets can be skipped, and only visible widgets get rendered. This can speed up the initial application start-up performance.
- Pause the refresh of chart and table after each click of user action. Users can take a couple of actions and then refresh the table/chat all at once.

To enable this capability, as an analytic application developer, you select Pause Data Refresh in the Builder Panel of the Table or Chart widget at design time.



Besides configuring the Table or Chart Builder Panel at design time, you, as an analytic application developer, can use the following scripting API to enable this capability at runtime as well:

```
Table.getDataSource().setRefreshPaused(paused: boolean);
Table.getDataSource().isRefreshPaused(): boolean;
Chart.getDataSource().setRefreshPaused(paused: boolean);
Chart.getDataSource().isRefreshPaused(): boolean;
```

You can also use the following API to enable and disable the refresh of several widgets at once:

```
Application.setRefreshPaused(dataSources: DataSource[], paused: boolean): void
```

When refresh is paused, end user interaction on a Table or Chart widget becomes meaningless. Most of the end user's interaction will take no effect until refresh is resumed. Therefore, it is useful to enable or disable interactions on a widget. You, as an analytic application developer, can use the following scripting API to enable or disable user interactions on the Table or Chart widget:

```
Table.setEnabled(enabled: boolean);
```

```
Table.isEnabled(): Boolean
Chart.setEnabled(enabled: boolean);
Chart.isEnabled(): Boolean
```

Scripting

The following best practices apply to scripting in analytic applications:

Group Several setVariableValue() Calls Against BW Live Connections

If you set several variable values with the method setVariableValue() of a data source, write these commands in one direct sequence, one after the other, without any other script methods in between. This sequence is folded internally into a single backend call to submit variable values instead of multiple ones, thus improving application performance.

If you use many widgets based on one model, and they aren't visible, at the same time, it can be better to use widget level variables to avoid implicit setting of variables on invisible widgets.

Prefer setDimensionFilter() over setVariableValue() with BW Live Connections

If a variable is affecting a dimension (for example, as a dynamic filter), consider using the method setDimensionFilter() of a data source instead of setVariableValue(). This avoids roundtrips to the BW backend server. However, variables are also used for other purposes than filtering. In such cases the usage of variables is still necessary.

Use getResultSet() Instead of getMembers()

The getResultSet method can use the resultset that is already available in the widget and does not need a roundtrip to the backend. The getMembers will always lead to an extra roundtrip to the backend system. So please consider using getResultSet if it is sufficient for your scenario.

If you need to use getMembers make sure to use the available options parameter to limit the list of returned members.

Avoid Changes in onResultChanged Event Script

Avoid changing the data source directly or indirectly in an onResultChanged event script. This might lead to an infinite loop.

Modifying Data Sources or Widgets at Application Startup

Configure the initial state of data sources or widgets (initial filter, feeding, variable values, and so on) at design time instead of using script methods during the onInitialization event. This event is executed after the widgets are initially loaded. Changing the state of the data source or a widget during this event results in another refresh, which may cause further roundtrips to the backend server.

If you need to modify data sources of widgets during the onInitialization event (by setting different filters or variable values), consider using the Pause Refresh API to pause the initial refresh of the application until it has been modified during the

application's onInitialization event. Proceed as follows: In the Builder panel of the widget's data source, enable Pause Data Refresh. In the onInitialization event script, modify the widget's data source. At the end of the script disable Pause refresh with setRefreshPaused(false).

Initialize Variables via URL Parameters

You can set (at the application level) variable values before the initial submit and the reception of the result sets by passing them as URL parameters of your analytic application's URL. This is the same functionality a story provides. For each variable you specify the model, variable, and variable value with the parameters v<number>Model, v<number>Par, and v<number>Val, where <number> is a two-digit number.

For more details see the help page Variable Parameters.

Note: In the example URLs of this help page replace the fragment story with application.

Example:

In the following example, the variable Manager with value ["SM1″,"SM2"] of model view:[_SYS_BIC][t.TEST][remotejuice] is passed via URL (the URL isn't functional out of the box as there are placeholders present for the tenant's SAP Analytics Cloud URL, the tenant ID, and the application ID):

https://<TENANT>/sap/fpa/ui/tenants/<TENANT_ID>/bo/application/<APPLICATION_ID>?v01Model=view:[_SYS_BIC][t.TEST][remotejuice]&v01Par=Manager&v01Val=["SM1″,"SM2"]

Avoid Repeating Instructions in Loops

Avoid repeating instructions in loops (for example, in for- or while-loops). Move these instructions before the loop. This applies also to instructions which seem to be cheap performance-wise, for example, Table_1.getDataSource().

Example:

```
// BAD
for (var i = 0; i < dimensions.length; i++) {
   Table_1.getDataSource().setDimensionFilter(dimensions[i], value);
}
// GOOD
var ds = Table_1.getDataSource();
for (var i = 0; i < dimensions.lenght; i++) {
   ds.setDimensionFilter(dimensions[i], value);
}
```

Prefer copyDimensionFilterFrom() over setDimensionFilter()

If you want to apply existing filters on several widgets, use the method copyDimensionFilterFrom() of a data source instead of setDimensionFilter().

Enable Planning on Tables Only when Planning is Used

If you add a table that is based on a panning model, then planning is enabled by default. If the table isn't visible at startup or if the table isn't used for planning at all, then disable planning. Use the method Table.getPlanning().setEnabled(true); of a data source to enable or disable planning, depending on whether you need planning or not, respectively.

Note that using the Pause Refresh capability is only possible if planning is disabled.

Use MemberInfo Object with setDimensionFilter()

If you use the method setDimensionFilter() of a data source and pass only a member ID, like in the following example, then a backend roundtrip is performed to fetch the member's description:

```
var memberId = Dropdown_1.getSelectedKey();
Table_1.getDataSource().setDimensionFilter("sap.epm:Department", memberId);
```

When you pass a MemberInfo object (it contains a description) instead of only a member ID string, like in the following example, then no backend roundtrip is performed:

```
var memberId = Dropdown_1.getSelectedKey();
var memberDescription = Dropdown_1.getSelectedText();
Table_1.getDataSource().setDimensionFilter("sap.epm:Department", {id: memberId,
description: memberDescription});
```

You can use this pattern also with an array of MemberInfo objects, like in the following example:

```
var resultSet = Table_2.getDataSource().getResultSet();
var memberInfos = ArrayUtils.create(Type.MemberInfo);
for (var i = 0; i < resultSet.length; i++) {
  var member = resultSet [i]["sap.epm:Department"];
  var memberId = member.id;
  var memberDescription = member.description;
  memberInfos.push({id: memberId, description: memberDescription});
}
Table_1.getDataSource().setDimensionFilter("sap.epm:Department", memberInfos);
```

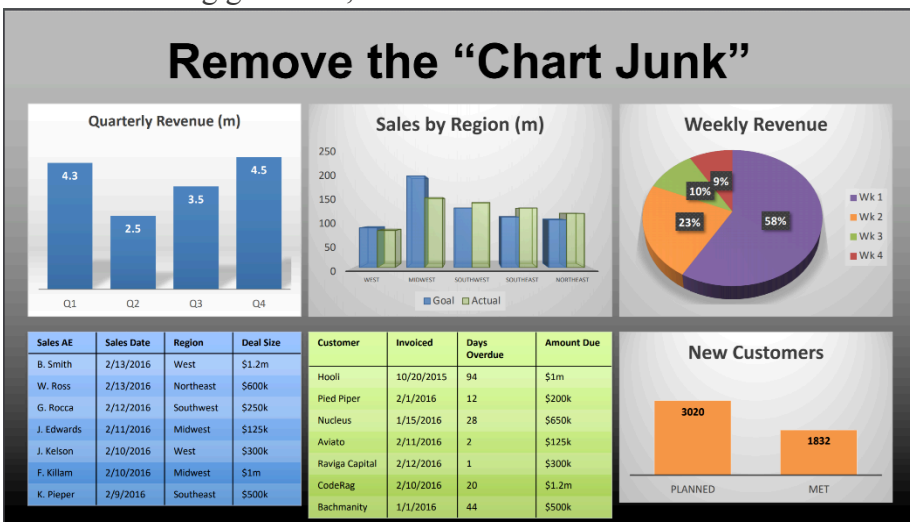
Apply this pattern whenever the member description is available to you. If the filter definition is never visible to the end user of your analytics application, you can even use a dummy description.

Improve Stories for Your Audience in SAP Analytics Cloud

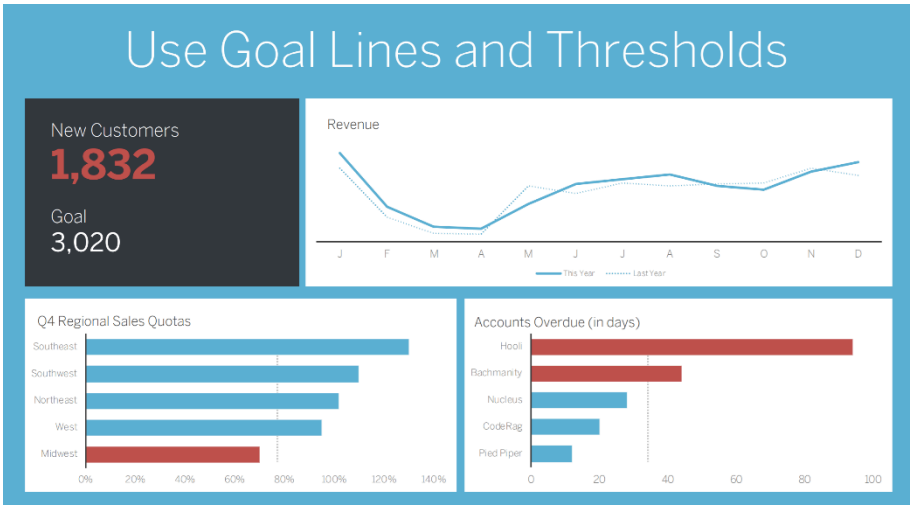
The prerequisite to the following 5 methods is to understand your audience – think about what data your end users may need and the simplest way you can provide this information.

1. **Follow the general best practices in data visualization**

- *Simplify your data visualization*. You may want to start off by removing charts which aren't very descriptive or telling. Your goal is to tell a story, so try to focus on the message you are trying to tell, while refraining from overwhelming your audience with too many visualizations.
- *Remove the "Chart Junk"*. Analyse your dashboard and remove parts of your chart that provide an unnecessary level of detail. Depending on your data, this may include removing grid lines, 3D effects and axis labels.



- *Use color mindfully.* Certain colors can mean something to your audience – red usually indicates a decrease in variance while green indicates an increase. We also want to use neutral colors that are not distracting to the human eye.
- *Add context.* Thresholds within charts can help your end users understand the data faster and jump to strong conclusions without spending too much time on interpretation. Descriptive titles also add value and help to reduce any audience confusion.

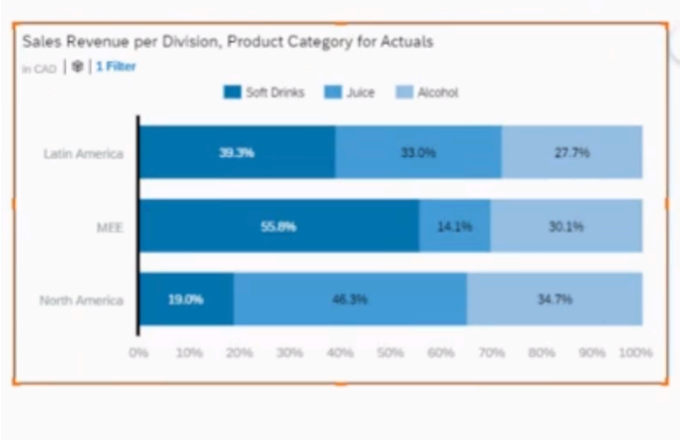


- To learn more about the best practices in data visualization, go to **5:48** minute of the webinar.

2. **Choose the appropriate chart type for your data**

- *Understand what you're trying to tell with your data.* For example, if you are trying to compare an element over a specific dimension, a bar chart will be easier for your audience to interpret. Similarly, if you are trying to display a core value, a numeric point will be interpreted faster than a chart. Discover more about understanding your data by visiting **8:55** and **15:10** of the webinar.

- *Pie charts are not very informative.* Due to the shape of a pie chart, your end users can misinterpret and overestimate the data. In place of a pie chart, a more detailed breakdown can be more insightful.
- *Use stacked bar charts, color, and leverage the built-in interactivity.* Stacked bar charts and the use of color help to split-up the data and make charts easier to read, without making any changes to the data itself. The built-in interactivity helps your end users compare different rows of data within a bar chart. To learn more, visit minute **34:00** of the webinar.

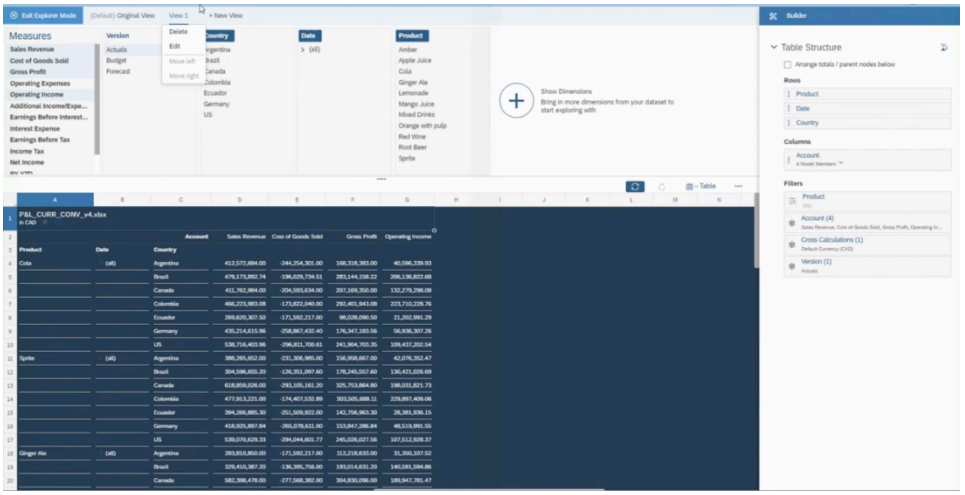


3. **Make your bar charts dynamic and enhance their visualization**

- *Create Input Controls.* Input controls hold a lot of storytelling power without making your dashboard too crowded. They allow your end users to analyse the data themselves. Think about your end users and what information may be useful to them as they're viewing your dashboard. Adding input controls is also helpful to introduce your end users to the dynamic capabilities in SAP Analytics Cloud. Learn more about input controls by visiting minute **42:40** of the webinar.

4. **Leverage the explorer view and beta table**

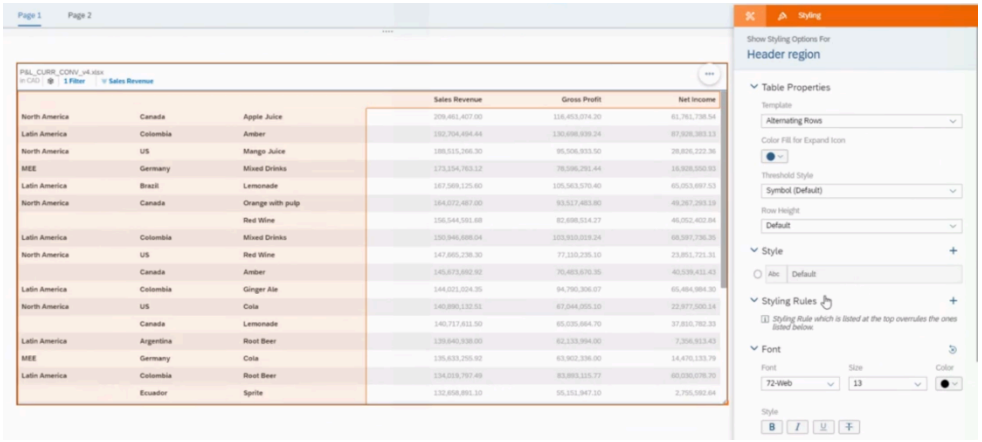
- *Use Explorer View.* Sometimes you need to add more detail into your charts without making your dashboard too messy. Explorer view allows you to add a granular level of detail into your charts without affecting your end users viewing experience and retaining the interpretability. Explorer allows you to create any table you'd like using different measures and dimensions.



- *Use Beta Tables.* Beta tables make your custom table easy to interpret using automatic formatting and offer performance improvements when loading large tables.
- Discover more about Explorer View and Beta Tables, visit **45:50** of the webinar.

5. **Using styling options to optimize performance**

- *Use regions when styling.* Many times, dashboard designers apply changes by highlighting columns or individual cells – this causes your styling rules to be applied cell-by-cell and decreases the performance of your dashboard. Instead, select a data region (for example, the header region) to apply style changes and improve performance.



- *Use hyperlinks.* Hyperlinks within your chart will help to add more granular detail for your end users without overwhelming them with too much information. You can apply filters to your hyperlinks to help your end users navigate through your charts more easily and access more detail.

Determine Your Goals

What Is Planning?

Reporting and analysis often focuses on historical data (known as actuals), but a business needs a clear vision of the future, too.

Planning is all about setting strategic goals and then determining how to meet those goals by creating annual budgets, tracking progress in forecasts, and simulating scenarios to find new opportunities. These plans are formed by projecting actuals into the future, by gathering input from different departments, and by considering trends, risks, and opportunities in the market.

Executive boards and finance departments play a big role in planning. But an effective plan needs input and support from the whole organization. Maintaining close integration between departmental plans and the overall strategic goals and financial plans is known as collaborative enterprise planning.

Roles Involved in Planning

A few different roles contribute to the process:

- **Planning modelers**: These specialists have a strong understanding of the organization's data and the other systems in their landscape. They'll play a big role in setting up the planning solution in SAP Analytics Cloud, including creating models and structured planning operations.

- **Planning reporters**: These users can be finance specialists who are working with the data on a daily basis to create budgets and forecasts, and to provide financial reports and analysis to business users and upper management.

Planning reporters also include many users who just need to contribute planning data for their specific area as part of a bottom-up planning process.

- **Planning viewers**: These are stakeholders in the planning process, such as business users and executives, who need to check to see how their team or organization is doing. Sometimes they might run their own simulations or contribute some planning data as part of a calendar task.

How Planning Fits into SAP Analytics Cloud
Applying analysis and predictive features while you're planning can help you plan faster and more accurately, and get a better understanding of your business.

For example:

- When starting a plan, you might use predictive scenarios to set the initial values, letting you quickly identify the overall trend as well as expected fluctuations in your data. The predicted values give you a quick start that's grounded in past performance.

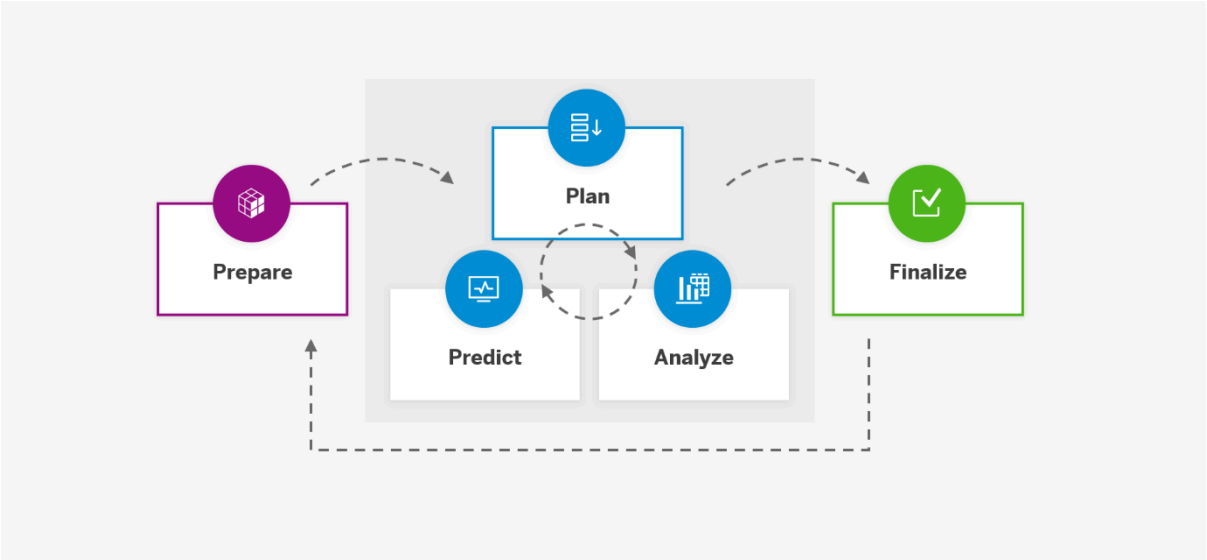
- While you're adjusting this plan with manual data entry, you might refer to overall KPIs as well as variance charts for each data point to get quick feedback about whether the plan is on target or not.

- If you want to understand the trends and drivers in your data, or investigate the root cause of a specific issue, you may want to do some free-form exploration of your data, and analyze it with smart features that identify key influencers and outlying data points.

You can integrate these features closely with the planning process so that data analysis, manual planning, and predictive features all reinforce each other without making your process slower or more complicated.

The Planning Process

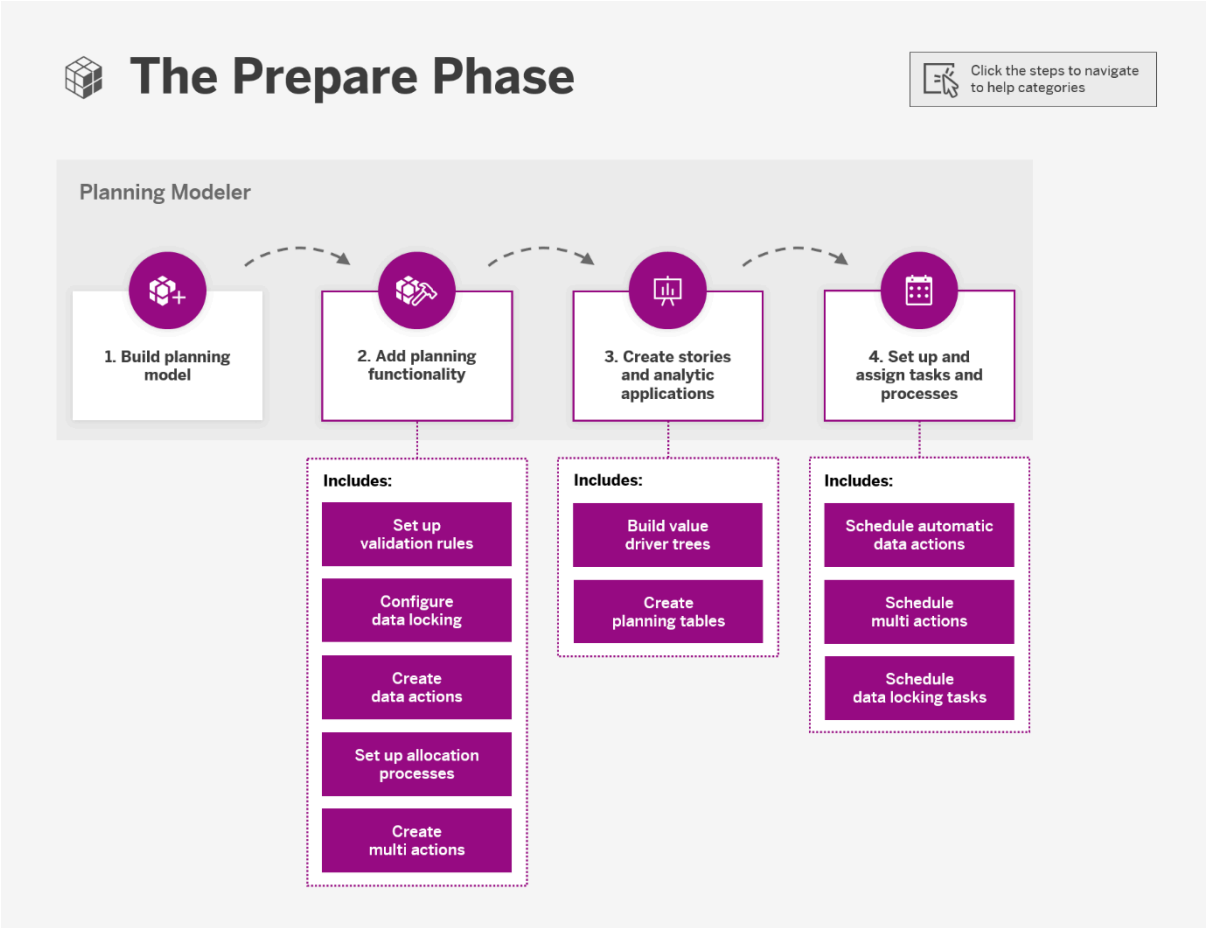
Planning processes are collaborative, cyclical, and iterative. Broad phases in a planning cycle include preparing the solution, planning, and finalizing the data:



Your organization might start a new cycle shortly afterwards, especially for rolling forecasts. You might also have simultaneous planning cycles running at different levels, such as an annual budget and quarterly forecast.

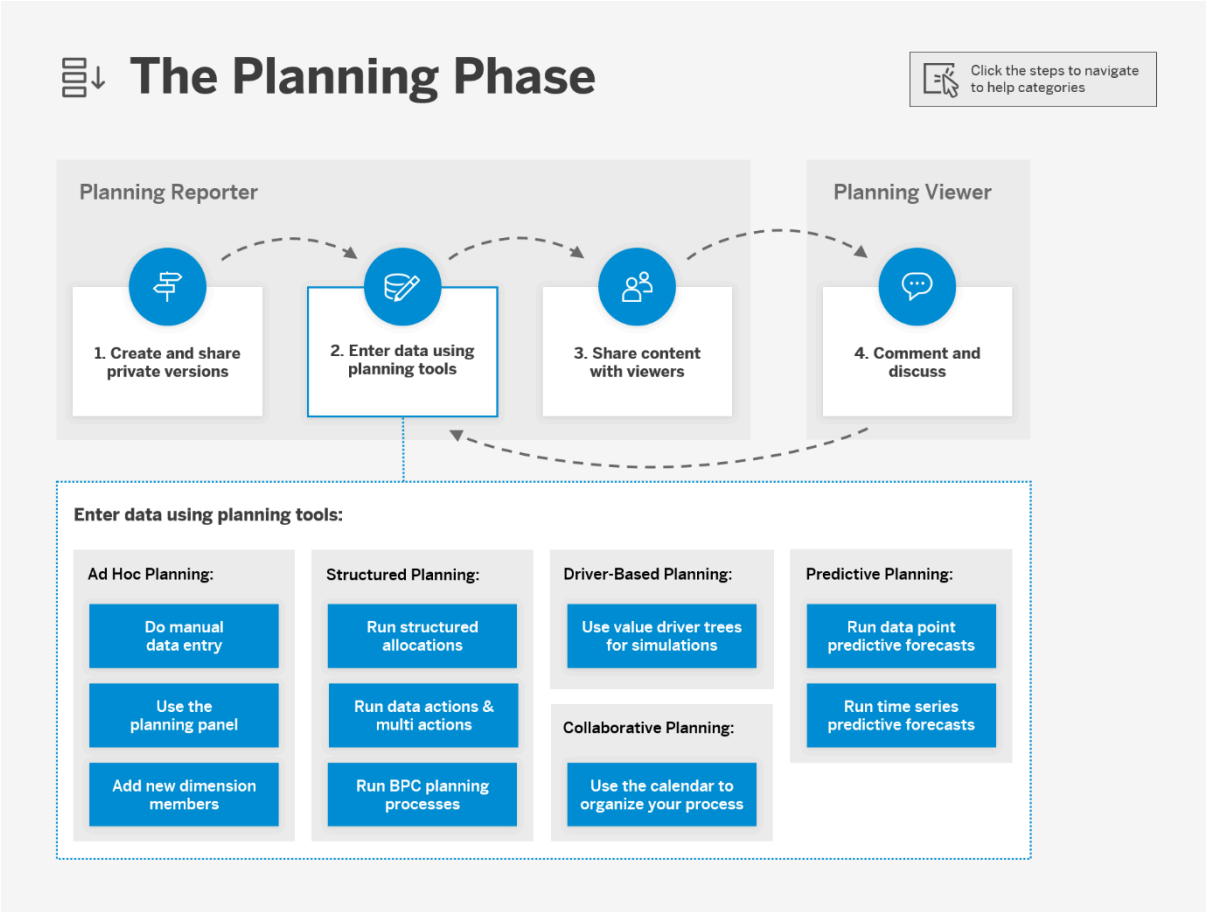
The Prepare Phase
Planning modelers do most of their work in this phase. After the initial setup of the solution, they can update and enhance it during later cycles.

Use this diagram to get descriptions of each step and click to open detailed instructions.



The Planning Phase
Next, planning reporters work to putting the figures into the plan, and sharing their work with viewers.

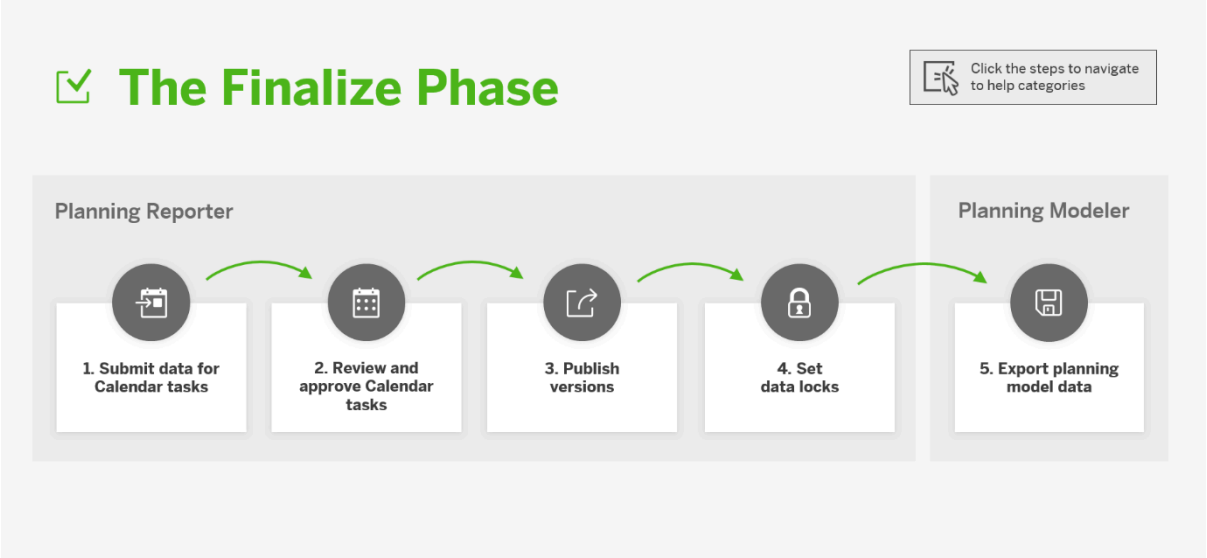
Use this diagram to get descriptions of each step and click to open detailed instructions.



There are often iterative changes in this part of the process as planners and stakeholders at different levels and in different departments collaborate on the plan.

The Finalize Phase

This image is interactive. Hover over each area for a description. Click highlighted areas for more information.



When the data is ready, there are a few steps to wrap up the process.

Other key features for planning workflows
Since SAP Analytics Cloud combines planning with analytics and predictive features, you can also support your planning workflows with several features that aren't specific to planning:

- Tables: You'll often use tables to keep track of your data, and several planning features are based here. While working with tables, you can also customize your layout and formatting and enhance the table with calculations, data point comments, and in-cell charts. For more information, refer to Use Tables to Visualize Data.
- SAP Analytics Cloud, add-in for Microsoft Office: If you prefer to work in Excel, you can use this add-in to connect to your SAP Analytics Cloud data, build tables, do data entry, and publish back to SAP Analytics Cloud. For more information, refer to SAP Analytics Cloud, add-in for Microsoft Office.
- Content library: SAP and its partners provide prebuilt content here, including best practices content for planning based on S/4HANA Cloud and integrated financial planning for SAP S/4HANA. Admins can explore, import, and adapt the content to get a headstart setting up their solution. For more information, refer to Getting Business Content and Samples from the Content Network.
- Collaboration tools: Basic collaboration tools let you share content and open discussions with other users. For focused discussion on specific data, you can use data point comments and commenting widgets. For more information, refer to Sharing, Collaborating, and Exporting.
- Currency conversion: For international business, you can do planning and analysis with multiple currencies. For more information, refer to Plan with Currency Conversion.

- Analytics designer: As a content creator, you might want to customize and streamline planning content to empower your end users. While stories are quicker and easier to set up, the analytics designer gives you more flexibility to create personalized analytic applications. For more information, refer to Analytic Application Design (Analytics Designer).

- Digital Boardroom: To help figure out the big picture, you can turn your stories into interactive presentations with touchscreen support. The ability to run simulations and explore the data helps your team make strategic decisions together. For more information, refer to Digital Boardroom Presentation.

Manage KPIs and Reports

Use
Manage KPIs and Reports app is a single platform for creating all analytical applications using KPIs, reports, multidimensional reports, and stories. You can create applications that can be launched directly from SAP Fiori launchpad. You can configure metrics in the KPI, visualize the data either in chart or table format, and analyze the data to improve the quantity and quality of the different business units belonging to an organization.

This app is integrated with SAP Analytics Cloud, which helps to analyze and interact with real-time data. With SAP Analytics Cloud, data can imported or connected live from SAP S/4HANA Cloud systems, modeled, and converted into various visualizations. This helps the business units in data-driven decision making. For more details, see the following section which explains the capabilities of Embedded Analytics powered by SAP Analytics Cloud.

SAP S/4HANA Cloud Embedded Analytics Powered by SAP Analytics Cloud
SAP Analytics Cloud is cloud-based analytics tool used by business organization units to analyze their data. You can create dashboards and reports in which you can find insights about your business operations. With seamless integration of SAP S/4HANA with SAP Analytics Cloud, you can:

- Create beautiful stories

- View integrated dashboards based on SAP S/4HANA Cloud data using SAP Analytics Cloud

- View and analyze stories with classic visualizations, navigate to them and share them with different business units in a secured environment

- Pull data from analytical and non-analytical data sources and analyze it at runtime

Disclaimer
When using the Smart Business functionality, you need to ensure that you adhere to local legal rules and regulations, for example, data privacy legislation. Neither SAP nor its associated companies assume any responsibility for the adherence to authorizations or data protection rules for data processed by its customers using Smart Business.

Prerequisites

- The **Analytics Specialist** role has been assigned to you.
- You must enable the app for extensibility. For more details, see Extensibility.
- The KPI content created in the quality system must be transported to production systems. For more details about transport, see Transporting Analytics-Based Extension Items. Only administrators can transport extensible items to a productive system using the **Transport Management** app.

Features
You can create groups, KPIs, reports, and stories from this page by choosing the relevant tabs. The basic functions, such as edit, copy, and delete, can be performed on the relevant entities by choosing the icons from the toolbar. The number groups, KPIs, reports, and stories are displayed for each entity. Using the Search tab, you can refine your search based on names, description, tags, and status.

The following features are available in the **Manage KPIs and Reports** app:

- **Groups:** You use this section to create groups for multiple KPIs.
- **Key Performance Indicators (KPIs):** KPIs are used to identify and measure the key metrics of a business. You create an evaluation to define a specific representation of a KPI, which means that you define a certain selection of the KPI data, the targets and thresholds that are relevant, together with some additional information. For example, you have defined a KPI to monitor car sales in a particular country. You can then create an KPI that restricts the results to a certain area of that country for which one sales representative is responsible. You must specify an active group when you are creating a KPI.
- **Reports:** You configure the reports for active KPIs as the starting point for further analysis. At runtime, you select the application to open either an SAP Smart Business generic drill-down application or a Data Analyzer and Story runtime. You can choose how to visualize the tile by selecting one of the available tiles (numeric, comparison, trend, or actual vs. target).
- **Multidimensional Reports**: This page displays all the applications for the released queries from the View Browser application and creating and launching of applications from these reports on the SAP Fiori launchpad.
- **Stories:** With the integration of SAP S/4HANA Cloud with SAP Analytics Cloud, you can create stories, add data, and generate powerful visualizations and analyze the data using **Data Analyzer and Story** runtime application.

Tell a story with your data

Preparing Data for a Model in a Story
When you initially acquire raw data, it may need to be cleansed, and it may include many rows and columns which might not be relevant to your analysis or planning. Data preparation involves pruning, transforming, and formatting your raw data before it's used in charts and tables.

Data preparation involves any of the following tasks:

- Fixing anomalies in the data (such as dates with inconsistent formatting), filling in empty cells, and so on.
- Filtering data, hiding columns, or splitting columns into two.
- Assigning dimension types or measures to the data columns, or setting columns to be dimension attributes.
- Mapping incoming data to existing dimensions, or mapping incoming numerical data to measures.
- Updating an acquired dataset by importing data or combining data from another source.

  For a detailed information on the basic preparing data experience see Combine Data with Your Acquired Data.

Refreshing a Model in a Story
You can refresh a private model inside a story by uploading another file. The contents of the uploaded file either replace, or are appended to, the existing content in the model. Follow these steps to refresh a model in a story:

1. With a story open in **Data** view (), select  (**Refresh Data**).
2. Choose a source file.
3. Choose whether the new data will replace, or be appended to the existing data.
4. Select **Finish**.

Publishing a Model in a Story
You can publish a model in a story to create a new public model based on the information in the private model. Reasons for publishing a private model include:

- If you want a public model that uses the same file and the same column settings as an existing private model, you don't need to create that public model from scratch.
- You may no longer have the original uploaded file, and therefore wouldn't be able to create a public model using that file.

Follow these steps to publish a model in a story:

1. With a story open in **Data** view (), select  (**Publish Model**).
2. Type a model name, and a description (optional).
3. Select **Create**.

A new public model, independent of the original private model, is created, and appears in the public models list on the **Models** page.

The original private model remains unchanged, and the story that contains it does not reference the new public model.

There may be differences between the public model and the original private model in the story:

- The dimension member IDs of a dimension in the public model can be different from the IDs in the corresponding dimension in the model inside the story. If you create a formula in the story that references dimension member IDs, the equivalent formula in the public model may refer to different dimension member IDs. However, in charts, both models will show the same dimension member values when you choose to display member descriptions instead of member IDs (dimension description is the default option for charts).
- Models in stories can have simple date dimensions (with drill down not enabled). After publishing, the corresponding dimensions in the public model would have dimension members represented as strings.

Replacing a Model in a Story
You can now replace any acquired model in your story with another compatible acquired model (replacing the classic account models with the New Model type is not supported). When you replace the model in your story, the configuration of all the widgets in your story is preserved. This means there's no need to rebuild your charts and tables.
Note

When a model is replaced in a story, there will be issues using the **Member Selector** for input controls.
Note

Model replacement uses identifiers for mapping both columns and dimension. Make sure the source and the target identifiers are the same. In some instances the IDs might be truncated and cause the model replacement to fail. To avoid truncation of IDs do not use the following for dimension or measure names:

- Names that contain more than 22 characters.
- Names that start with a number.
- Names that contain special characters.

Your target model doesn't have to be the same model type (planning or analytic) as the original model in your story. You can replace an analytic account model in your story with a planning model, and vice versa. Your new model must contain all the measures and dimensions, and the same measure and dimension names and types as your original model. However, your new model can also contain additional measures and dimensions not found in the original model.

Follow these steps to replace a model in a story:

1. Open a story in **Data** view ().

2. Select beside your current model name.
3. Select to open the **Select Model** dialog.
4. In the (Search) field, type the name of the new model with which you'd like to replace your current model, and select it from the list.

   Note

   Models identified as not compatible will be hidden and not displayed in the **Select Model** dialog.
5. A **Replace Model** dialog appears informing you about the potential differences in your story after replacing the model. Select **OK**.
6. Your model is now replaced.
7. Select (Save) to save your story based on the new model.

Note

- It is currently not possible to replace the classic account type model with the newer New Model Type and vice versa.
- If you want to base your **Smart Discovery** results on your new model, you need to run **Smart Discovery** again, from the same story.
- If you replace a model using the **R** visualization widget, the original model name persists. Only the model ID is replaced.

Provide context

Context dependent Data Actions in SAP Analytics Cloud

Context is everything" is a sentence often heard and rather fitting in many scenarios. For example: would you think a government lockdown is acceptable? In 2019 you might not, in the current context of COVID-19 you might think otherwise. Whatever your stance on this and many other topics, your answer will be dependent on the context in which it is given.

In relation to planning operations many of them will also be context dependent. Bringing us to the topic of my blog: Context dependent Data Actions which are now available in the preview release of SAP Analytics Cloud. This allows the user to run business logic on the slice of the planning model that they are currently focusing on, i.e. In context.

Functionality

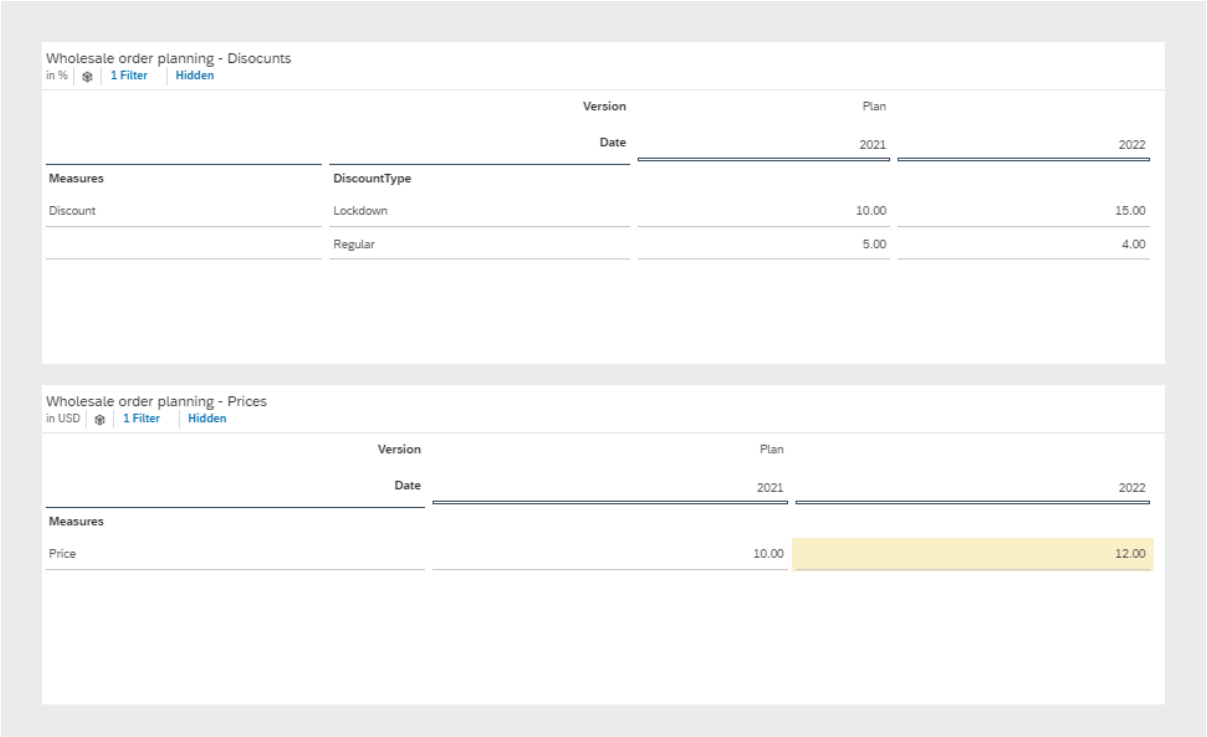
The new context sensitivity is actually building on pre-existing functionality in SAP Analytics Cloud called parameters. Using parameters one could already provide context to a data action. More information on the topic of using parameters can be found here.

What is new is that these parameters can now also be read directly from story- and page-filters, thus the context of the Story which the user is in at that point in time, let's take a look at a simple example.

Example case

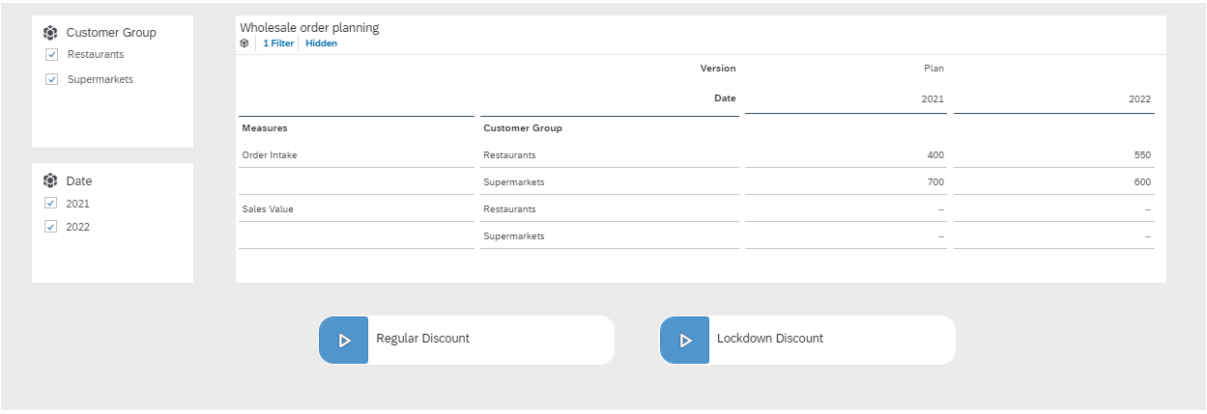
We'll be looking at a wholesale company selling to two types of customers: Supermarkets and Restaurants. In their planning process they have one planner looking at price and discounts independent of customer group. Subsequently for order intake planning they will be taking customer group into account and derive based on the prior planned prices and discounts the total expected sales value.

In our first step our planner decides to run a small price increase year over year following inflation but has a different discount percentage based on the context being in a lockdown or not. Providing significantly higher discounts in the lockdown situation.



Price and Discount Planning

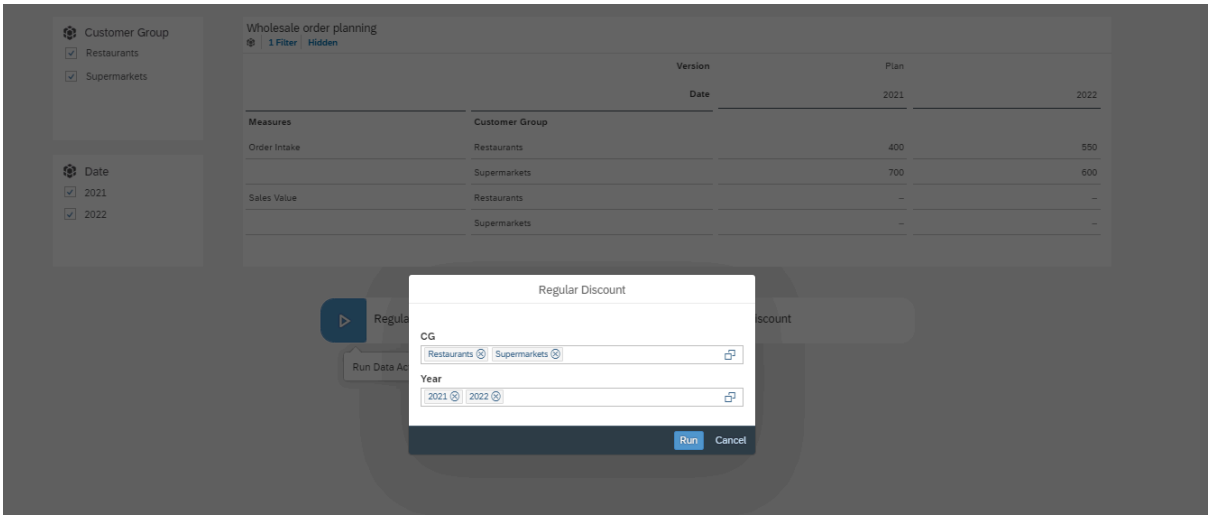
In the subsequent planning step Order intake values are planned by customer group, where page filters for Customer Group and Date can help planners zoom in on their desired context.



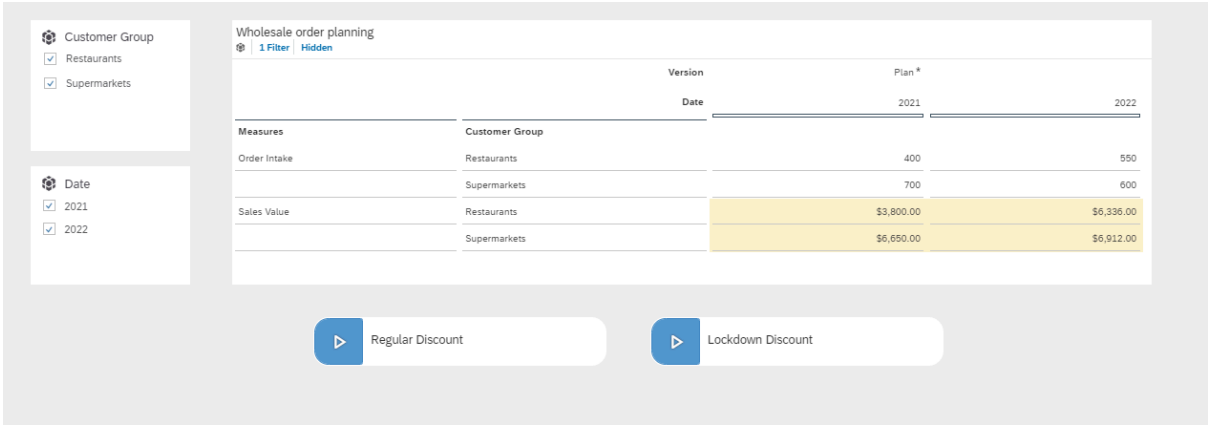
Order intake planning

The two Data Actions below will calculate the Sales Value on either the regular discount, or the defined lockdown discount level.

Our planner first runs the regular discount data action to come to the preliminary Sales value. The context in this case is the complete picture as can be seen in the member selector pop-up based on set filters.

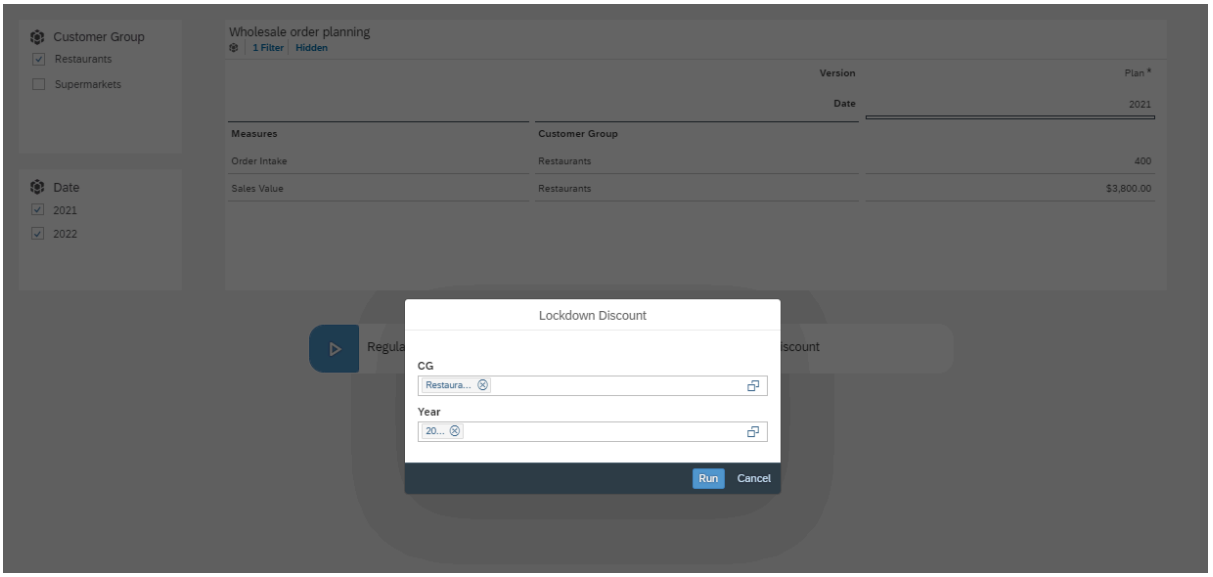


Regular discount data action in context

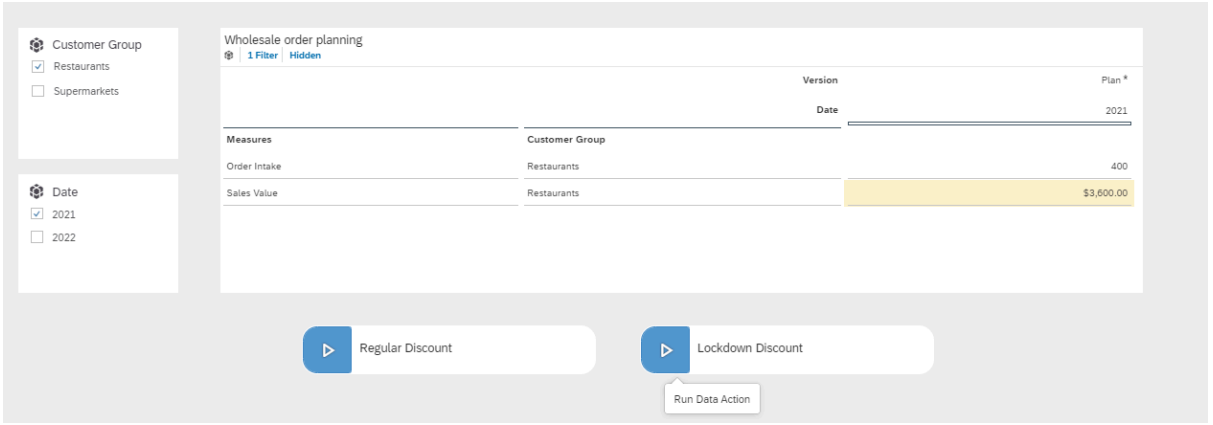


Results

Our planner however feels that given the lockdown will most probably endure for the Restaurant business in 2021, it would be best to apply the lockdown discount in 2021 for the Restaurant business. Therefore, the filter is set to Restaurant and 2021 and the Lockdown data action will be triggered. Context for the data action is now limited to the settings of the page filters as can be seen in the member selector pop-up.

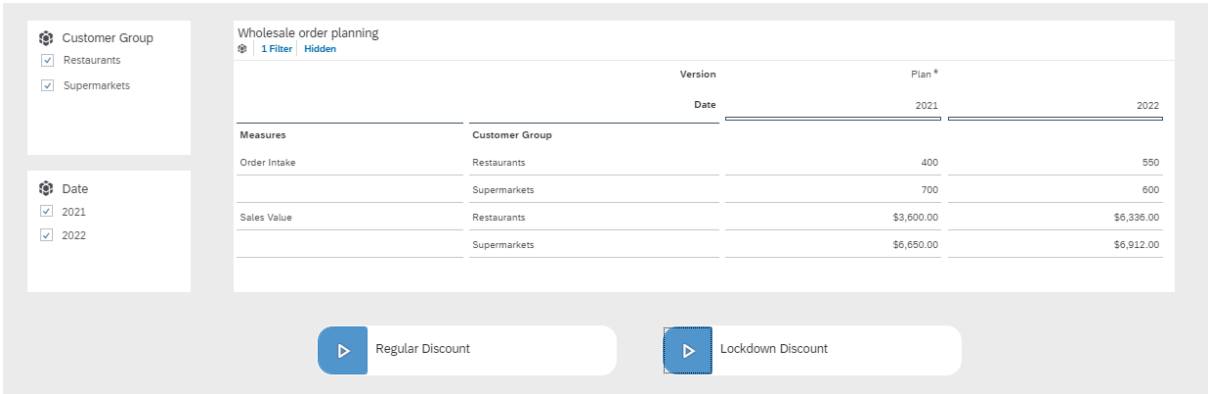


Lockdown discount data action in context



Results

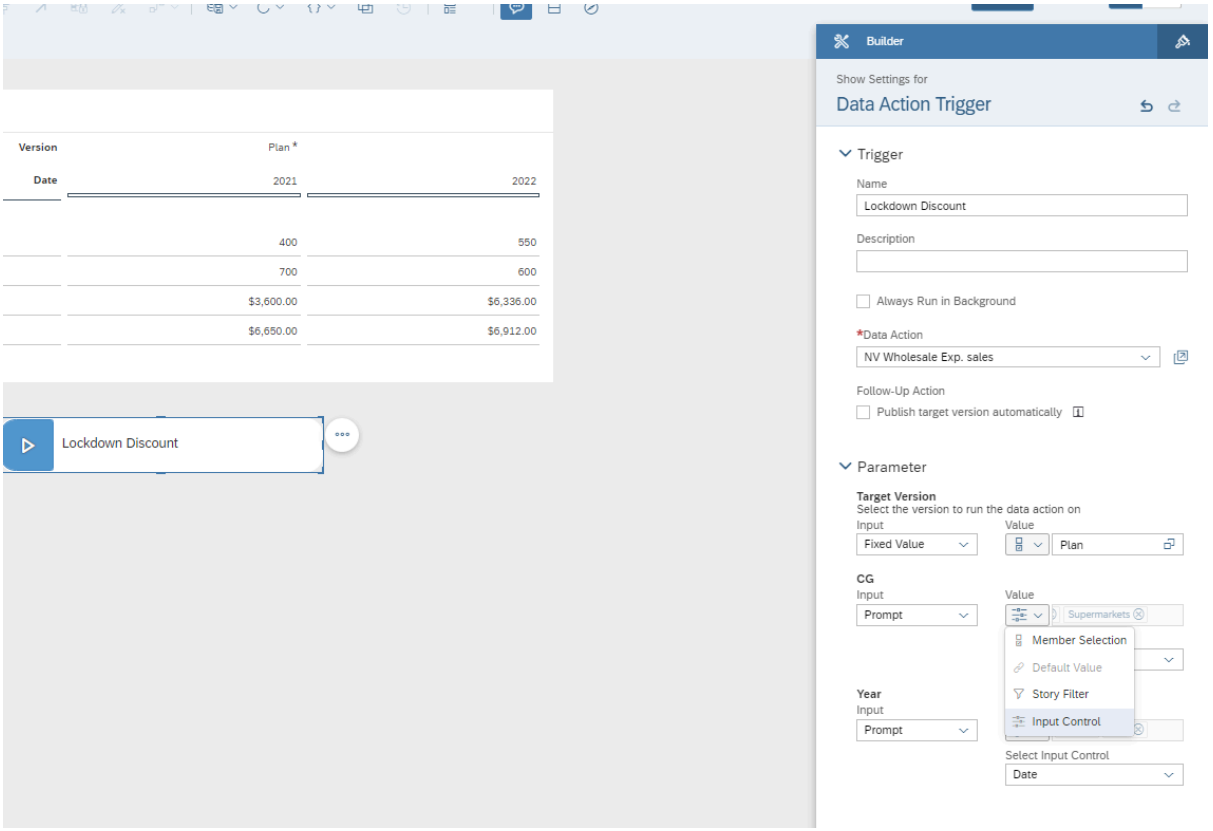
Zooming out again using page filters again shows the full picture after applying the data actions in their respective contexts.



Overall view

How to setup

Within the data actions two parameters are defined as explained before. The new functionality is that the entry of these parameters can be done using page- and story-filters. This can be setup in the data action trigger in the story like below.



Data action trigger settings – context setup

Benefits

The benefits are twofold in my mind, from a business point of view you provide the end-user with a logical selection for the data action in context of their current view. On top of that you are increasing overall performance by limiting the scope of the data action to where it is relevant.

Pall the right information on the page

Convert Between Story Filters and Page Filters
You can convert between story filters and page filters.

If you have a story filter and you want the filter to apply only to the charts on one story page, you can convert the story filter to a page filter.

The workflow for converting a story filter to a page filter involves the following steps:

1. With a canvas or responsive page open, select a story filter from the filter bar.
2. Select **Convert to Page Filter**.

The new filter appears as an object on the canvas or responsive page.

You can resize the filter object by selecting it and dragging its sizing handles. If you enlarge the filter object, it becomes an input control that you can use to select filter values.

Example

If the filter is set to allow viewers to change the filter values, and to allow multiple filter values, you can enlarge the filter object on the page so that the filter values appear in a list, with checkboxes. Then you can change filter values by selecting and deselecting the checkboxes.

Tip

If you want to convert a page filter to a story filter, you can select the page filter and choose **(More Actions) Convert to Story Filter**. The story filter will apply the filter to all pages in the story.

Measure-Based Filters

Measure-based filters are filters that are based on a range of values in a measure. For example, if you want to include in your story only your company locations that have more than 200 employees, you could create a filter based on an Employees measure.

For more information, see Measure-Based Filters.

Effects of Multiple Filters (Cascading Effect)

Changes you make to a story or page filter affect related filters in the same story or page. (Story filter changes affect both story filters and page filters in the story.)

Tip

Story filter changes cascade down to page filters and page filter changes cascade down to group filters, but no filter changes cascade up.

However, when multiple group filters are configured to affect the same set of widgets through linked analysis, the cascading effect is applied to all the group filters.

For more information on group filters, see Group Filters.

For example, when you change a page filter value, any related page filters on the same page are updated automatically. If you have both Country and Region filters on a page, and you change the Country filter value from All to Sweden, the Region filter updates to show only regions within Sweden. All other region names are hidden. You can select **Show Inactive Values** to display those hidden values.

Restriction

When an input control with the **Show unbooked members** option turned on has filters cascading their changes into it, the unbooked members won't be included in the input control's list of members.

Note

Some filters are not affected by other filter changes:

- Range filters
- Hyperlink filters
- Multi-dimension filters (filters that apply to a combination of dimensions)

You can turn off the cascading effect in the settings for the filter or input control. When the cascading effect is turned off, any filter value selection changes made in this filter do not affect other filters, and any filter value selection changes made in other filters do not affect this filter.

Date and Time Range Filters
You can define time periods based on years, half-years, quarters, months, weeks, or days (depending on the time granularity defined in the underlying model), and apply the date range as a filter, so that only details in the selected time period are visible.

If your data is recorded very frequently; for example, sensor data, you could use the Timestamp dimension type, and be able to define time range filters based on hours, minutes, seconds, and milliseconds.

The ranges can be fixed or dynamic; for example, you could choose the fixed range January 2019 to December 2019. If this story is opened in 2020, the story will still show 2019 data. Dynamic date ranges shift based on the current date. They offer a few more granularities such as current year, current quarter, and current month, and you can offset the range from the current date.

Examples of dynamic date ranges:

- If you want to display data from three years ago, to two years into the future, you would choose **Year** granularity, enter 3 under **Look Back**, and enter 2 under **Look Ahead**. If the current year is 2019, then the date range is 2016 to 2021.
- If you want to display data for the current quarter, you would choose **Current Quarter** granularity. If the current date is June 1, 2019, then the date range is April 1, 2019 to June 1, 2019.
- If you want to shift the entire range forwards or backwards instead of basing it around the current date, choose **Offset** as the range type. For example, to set the range as the year before the previous year, you choose **Year** granularity, select **Look Back** as the offset direction, **Year** as the offset granularity, and 2 as the offset amount. If the current date is June 1, 2019, then the date range is 2017. Note that the **Offset Granularity** can’t use a shorter period than the overall **Granularity**.
- If you switch on **Include Range up to Current Period**, the date range ends at the current period. If you choose **Year** granularity, and enter 3 under **Look Back**, and the current date is June 1, 2019, then the date range is January 1, 2016 to June 1, 2019. For offset ranges, the **Include Range up to Offset Period** setting works the same way using the offset period instead of the current period.

Note

Offset ranges aren't available for filters applied to maps.

It is also possible to define multiple range time filters and apply these together. You could use this, for example, to compare the first two months of the year over a three year period by defining three separate ranges for months Jan–Feb for each of the three years. When these ranges are applied as a single filter, everything else except the selected periods is filtered out.

In some workflows, for example, planning, you might want to set the current date of a dynamic date range filter to be different from today's actual date. To learn how to do this, see Customizing the Current Date.

Note

- You can have multiple date dimensions in your story.
- You may want to restrict the date ranges that viewers of the story can choose. For example, if you don't want viewers to be able to select future dates, disable the **Look Ahead** field by selecting ⋯ 👁 **(Show/Hide)**, and clear the check mark beside **Look Ahead**.
- For fixed date range filters, if you define a range using the Day granularity, all other ranges in the filter can have only the Day granularity. The same restriction applies to Week. However, the Year, Half-Year, Quarter, and Month granularities can be mixed. This restriction doesn't apply to dynamic date range filters; for example, you can define one range with day granularity and another range with year granularity.
- If you choose Day granularity, you can manually type dates into the date fields. If you do type in dates manually, you'll need to use the date format defined by the **Date Formatting** setting in your user profile. Dates entered in any other formats won't be recognized as dates.
- Time Range filters work for booked data only.

Static vs. Dynamic Filters
When you create a filter (a story filter, page filter, or calculation input control) for a flat dimension, if you select “All Members”, a dynamic filter is created. This means that the latest dimension member descriptions are always fetched from the model. But if you select individual dimension members, and not “All Members”, a static filter is created. This means that the dimension member descriptions are remembered from the time when the filter was created.

Example: Say you create a filter with a list of countries, and initially the list of countries is: Canada, USA, and Germany. If you create a static filter, those three full country names will be remembered. Later, someone changes the dimension member descriptions in the model or the back-end server to CA, US, and DE. The static filter will still display Canada, USA, and Germany, instead of CA, US, and DE. A dynamic filter though, will display the updated member descriptions whenever the story is opened.

Story Filters in the Optimized Story Experience
To access the story filters that are not currently displayed in the story filter bar, select **See More Filters**, and then select a filter from the dropdown list.

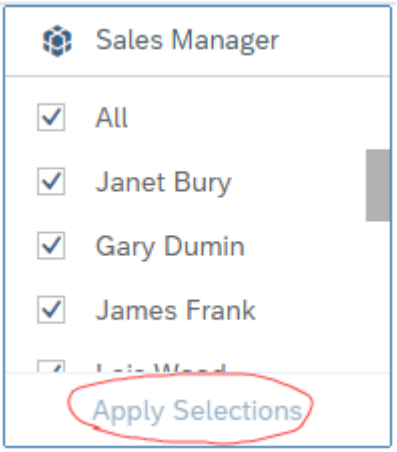
Restriction

Some story filter options (such as converting between story and page filters) are not yet supported.

Modifying Member Selections

As a story designer, you can decide how your users apply changes to multi-selection filters:

- Immediately update the chart or table contents as members are selected or deselected in the filter.
- Make your changes in the filter dialog and then use **Apply Selections** to apply all those changes at the same time.



This is the default setting for new member filters that allow multiple selections.

You can choose which method to allow for each filter by showing or hiding the option **Apply Selection Button**.

- Story filters: select the filter and then select **Show/Hide Apply Selection Button**.
- Page filters (input controls): right-click the filter and then select **Show/Hide Apply Selection Button**.