



SAP Analytics Cloud, analytics designer

Developer Handbook

Document Version: 12.2 – 2023-05-29

What's New

Version 12.2

Updated add more information of Saving a Snapshot

Version 12.1

New Input Control API

Updated Publishing and Reverting Data Changes

add Publish and Leave Dialog for Planning

Updated Analytic Application Script Performance Popup

New Use Application.refreshData() to refresh multiple data sources

Updated Use the Pause Refresh API

add Pause each widget individually for a group of widgets

New Keeping Last Saved Values of Dynamic Variables in Bookmark

Version 12.0

New Generic Widget Access API

New Blend Data in Analytic Applications

Version 11.3

Updated Minor typographic changes

Version 11.2

New Tip: Consider Device Viewport When Using Loading Widgets in Background

New Tip: Loading Bookmarks in the onInitialization Event Script

Updated Members-on-the-Fly
createMembers() with members with the same ID results in an error

Updated Bookmark API
rearranged

Updated Minor typographic changes

Removed Table of Figures

Version 11.1

Updated Minor typographic changes

Version 11.0

New Getting Information About a Data Source

Updated Pause Refresh

New Using the canUserSubmit API

New Using the canUserApprove API

New Using the canUserReject API

New Using the canUserDecline API

Table of Contents

1	About Analytics Designer	1
1.1	What Is an Analytic Application?	1
1.2	What Is Analytics Designer?	1
1.3	What Can You Do with Analytic Applications That You Can't Do with Stories?	1
1.4	How Are Stories and Analytic Applications Related to Each Other?	1
1.5	Why Do We Need Both Stories and Analytic Applications?	2
1.6	What Is the Typical Workflow in Creating an Analytic Application?.....	2
1.7	What Are Typical Analytic Applications?	3
1.8	How Does Scripting Work in Analytic Applications?	3
1.9	What's the Scripting Language for Analytic Applications?	4
2	Getting Started	5
2.1	Prerequisites	5
2.1.1	Required Access.....	5
2.1.2	Required Roles	5
2.1.3	Required Licenses	5
2.1.4	Modes	6
2.2	Designing Elements	6
2.2.1	Canvas.....	6
2.2.2	Widgets and Filters	6
2.2.3	Data Sources and Models	6
2.3	Managing Your Analytic Application	7
2.3.1	Transporting an Analytic Application	7
2.3.2	Sharing an Analytic Application	7
2.3.3	Bookmarking Your Analytic Application	7
2.3.4	Translating Your Analytic Application	8
2.3.5	Exporting Analytic Application Content.....	8
3	Designing an Analytic Application.....	10
3.1	Creating an Analytic Application	10
3.2	Browsing for an Analytic Application.....	10
3.3	Opening Analytic Applications in a Specific Mode.....	11
3.3.1	Opening an Analytic Application from File Repository with CRUD Permissions	11
3.3.2	Opening an Analytic Application from File Repository with Read Permissions	11
3.3.3	Opening a Mode with the URL.....	11
3.3.4	Switching Between Present and View Mode	12
3.4	Saving and Leaving Analytic Applications	12
3.5	Toolbar Functionalities	13
3.5.1	Toolbar in Edit Mode.....	13
3.5.2	Toolbar in View Mode	13
3.6	Edit Mode Functionalities	13
3.6.1	Outline and Side Panels	13

3.6.2	Scripting Section	14
3.6.3	Layout Section	15
4	Scripting in Analytics Designer.....	19
4.1	Why Scripting?	19
4.2	Scripting Language Overview	19
4.2.1	Type System	19
4.2.2	Tooling – Code Completion and Value Help	19
4.2.3	Events	19
4.2.4	Global Script Objects	20
4.2.5	Accessing Objects	20
4.2.6	Script Variables.....	20
4.3	Script Editor.....	21
4.3.1	Creating and Editing Event-Based Scripts.....	22
4.3.2	Creating and Editing Functions in Global Script Objects.....	23
4.3.3	Script Editor Layout.....	24
4.3.4	Keyboard Shortcuts	25
4.3.5	Info Panel: Errors and Reference List.....	25
4.3.6	Renaming Widgets, Script Variables, and Script Functions	25
4.4	Scripting Language Features.....	25
4.4.1	Typing	25
4.4.2	No Automatic Type Casting	26
4.4.3	Accessing Objects	26
4.4.4	Finding Widgets with Fuzzy Matching	26
4.4.5	External Libraries	27
4.4.6	Debugging with console.log()	27
4.4.7	Loops	27
4.4.8	Double and Triple Equals Operators	28
4.4.9	if and else Statements	29
4.4.10	this Keyword	29
4.4.11	switch Statements.....	29
4.4.12	break Statement.....	29
4.4.13	Debugging Analytics Designer Scripts in the Browser	30
4.4.14	Using Two-Dimensional Arrays	33
4.5	Data Sources and Working with Data	35
4.5.1	Setting Range and Exclude Filters	35
4.5.2	Getting Dimension Filters	37
4.5.3	Dimension Properties.....	38
4.5.4	Hierarchies	38
4.5.5	Getting Members	39
4.5.6	Getting Information About a Data Source.....	42
4.6	Working with Pattern-Based Functions.....	43
4.6.1	Adding a Pattern-Based Function	43
4.6.2	Creating the Pattern.....	43
4.6.3	Scripting	44
4.6.4	More Examples	44
4.7	Method Chaining	44
4.8	Script Runtime	44

4.9	The R Widget and JavaScript	45
4.10	Differences Between SAP Analytics Cloud and Lumira Designer	46
5	Widget Concepts, Script APIs, and Usages	47
5.1	Basic Widget Concepts.....	47
5.1.1	Supported Widgets	47
5.1.2	Custom Widgets.....	47
5.2	Builder Panel.....	48
5.3	Styling Panel	48
5.4	Action Menu	49
5.5	Script Editor View.....	49
5.6	Table	50
5.6.1	Table API	50
5.6.2	More Table APIs	52
5.6.3	Table Events	53
5.6.4	Formatting Numbers	53
5.7	Chart	54
5.7.1	Chart API	54
5.7.2	Chart Events	56
5.7.3	Formatting Numbers	56
5.8	Result Set API.....	57
5.8.1	Using the getResultSet API	57
5.8.2	Using the getDataSelections API.....	74
5.8.3	Using the getResultMember API	76
5.9	Widget Level Refresh Data API	80
5.10	Prompt API.....	80
5.10.1	Opening the Prompt Dialog	80
5.10.2	Getting Variables	81
5.10.3	Setting Variable Values.....	81
5.10.4	Getting Variable Values	83
5.10.5	Removing Variable Values	84
5.10.6	Copying Variable Values	84
5.11	Popup and Dialog	85
5.11.1	Main Popup and Dialog API.....	85
5.11.2	Button-Related Popup and Dialog API	86
5.11.3	Popup and Dialog Events	86
5.11.4	Known Restrictions of Popup and Dialog	86
5.12	Text Widget	87
5.12.1	Changing Text.....	87
5.12.2	Adding Dynamic Text.....	87
5.13	RSS Feed.....	87
5.14	R Visualization	88
5.15	Geo Map	89
5.16	Timer	89
5.16.1	Script API	89
5.16.2	Sample 1 – Create Animation	90

5.16.3	Sample 2 – Automatically Play the Application	90
5.17	List Box	91
5.17.1	Creating a List Box.....	91
5.17.2	Building a List Box	92
5.17.3	Configuring a List Box.....	92
5.17.4	Events	93
5.17.5	Script API.....	93
5.17.6	Sample Use Case	94
5.18	Layout API.....	95
5.19	Set Theme API.....	97
5.20	Loading Indicator	98
5.20.1	Automatic Loading Indicator	98
5.20.2	Script API	99
5.21	Bookmark API	99
5.21.1	Saving Bookmarks	99
5.21.2	Saving Additional Properties with Bookmarks	99
5.21.3	Keeping Last Saved Values of Dynamic Variables in Bookmark	101
5.21.4	Loading Bookmarks	102
5.21.5	Getting Bookmark Information	102
5.21.6	Deleting Bookmarks.....	103
5.22	Notification API	103
5.23	Move Widget API	105
5.24	Property Binding of Simple Widgets	106
5.24.1	Binding the Value of a Simple Widget	106
5.24.2	Enabling Write-Back at Runtime	108
5.25	Get Application Info API.....	109
5.26	Application Messages API	109
5.27	Gestures on Mobile Devices	110
5.28	Set Widget Style API.....	110
5.29	Value Driver Tree	111
5.30	Pause Refresh	115
5.30.1	Builder Panel.....	115
5.30.2	Script API	117
5.31	Data Explorer API	118
5.31.1	Showing or Hiding Additional Dimensions when Opening Explorer	118
5.31.2	Configuring the Order of Additional Dimensions	118
5.31.3	Applying Explorer Results to a Chart or Table	118
5.32	Flow Panel	119
5.32.1	Fixed Area and Flow Container	119
5.32.2	Responsive Rule Configuration	120
5.33	CSS Support	121
5.33.1	Supported Class Names and Properties	121
5.33.2	Creating the Application CSS	122
5.33.3	Using the CSS Class in a Widget	122
5.33.4	Setting the Global Default Class Name	123

5.33.5	Loading CSS into a Theme.....	125
5.34	Calendar Task API	125
5.34.1	Using the Calendar Integration Component	125
5.34.2	Using the getCurrentTask API	127
5.34.3	Using the getStatus API.....	127
5.34.4	Using the getType API.....	128
5.34.5	Using the hasUserRole API	128
5.34.6	Using the submit API	128
5.34.7	Using the decline API.....	129
5.34.8	Using the approve API	129
5.34.9	Using the reject API	130
5.34.10	Using the getCalendarTaskById API	130
5.34.11	Using the activate API.....	131
5.34.12	Using the getRelatedTasksId API.....	131
5.34.13	Using the createCompositeTask API.....	131
5.34.14	Using the Status Change APIs for any Calendar Composite Task	132
5.34.15	Using the canUserSubmit API	132
5.34.16	Using the canUserApprove API	133
5.34.17	Using the canUserReject API	133
5.34.18	Using the canUserDecline API	134
5.35	Data Actions.....	135
5.35.1	Prerequisites	135
5.35.2	Adding a Data Action Component	135
5.35.3	Script API.....	136
5.36	Comments	137
5.36.1	Working with Comments	137
5.36.2	Comment Widget	138
5.37	Export API	141
5.37.1	Export to PDF	141
5.37.2	Export to Excel.....	145
5.37.3	Export to CSV	146
5.38	API for Models with Accounts and Measures	148
5.38.1	Chart	148
5.38.2	Data Explorer	148
5.38.3	Smart Discovery.....	149
5.38.4	Smart Grouping.....	149
5.39	Open in New Story API	149
5.39.1	Script API.....	150
5.40	Navigation API	150
5.41	Web Page	151
5.41.1	Using Sandbox Restrictions.....	151
5.42	Dialogs to Share Applications and Bookmarks.....	152
5.42.1	Customize Link to Shared Application or Bookmark	152
5.42.2	Script API	154
5.43	Generic Widget Access API.....	154
5.44	Input Control API.....	155

6	Typical Patterns and Best Practices.....	156
6.1	Switching Between Chart and Table.....	156
6.2	Selecting Measures via Dropdown or Radio Button to Filter Table and Chart to Display (Single Selection)	160
6.3	Selecting Measures via Dropdown to Filter Table and Chart to Display (Multi-Selection)	167
6.4	Using Filter Line for Filtering Table, Chart, and R Visualization	177
6.5	Cascaded Filtering	183
6.6	Adding and Removing Dimensions in Rows and Columns for Table	190
6.7	Creating a Settings Panel Using a Popup Window.....	211
6.8	Selection Handling in a Table or Chart and Open a Details Popup.....	230
6.9	Using R Widget Word Cloud for Visualization	252
6.10	Setting User Input for Planning Data	272
7	Planning	274
7.1	What to Expect from Analytics Designer Regarding Planning?.....	274
7.2	Basic Planning Concepts in Analytics Designer	274
7.3	Refreshing Your Data	276
7.4	Setting User Input	276
7.5	Planning Versions	277
7.5.1	Private Versions.....	277
7.5.2	Public Versions	277
7.6	How to Manage Versions.....	278
7.6.1	Publishing and Reverting Data Changes.....	278
7.6.2	Copying	280
7.7	Data Locking	281
7.7.1	Using <code>getDataLocking()</code>	281
7.7.2	Using <code>getState()</code>	282
7.7.3	Using <code>setState()</code>	283
7.8	Planning Events	283
7.8.1	<code>BpcPlanningSequence</code>	283
7.8.2	<code>DataActionTrigger</code>	284
7.9	Members on the Fly	284
8	Predictive	286
8.1	Time Series Forecast	286
8.1.1	Switching Forecast On and Off	286
8.1.2	Configuring Forecast.....	286
8.2	Smart Insights	287
8.2.1	Discover per Selected Data Point.....	287
8.3	Smart Grouping.....	288
8.3.1	Switching Smart Grouping On and Off	288
8.3.2	Configuring Smart Grouping	289
8.4	Smart Discovery	289
8.5	Search To Insight	290

9	OData.....	294
9.1	What You Should Know About OData	294
9.2	How You Can Connect to OData.....	294
9.2.1	What You Need to Do	294
9.2.2	Known Restrictions	294
9.2.3	What Is an Action?	295
9.2.4	What Are Action Imports?.....	295
9.2.5	What Is a Bound Action?	295
9.3	How You Can Call OData Actions	296
9.4	How You Can Read Data from OData Services	302
9.4.1	Retrieving a Single OData Entity	302
9.4.2	Retrieving All Entities from an OData EntitySet.....	303
9.4.3	Retrieving Specific Entities from an OData EntitySet.....	303
9.4.4	Retrieving the Number of Entities from an OData EntitySet.....	303
9.4.5	Known Restrictions	304
10	Post Message API.....	305
10.1	Scenario 1: How You Can Embed an Analytic Application in a Host HTML Page via iFrame	305
10.1.1	postMessage.....	305
10.1.2	onPostMessageReceived	306
10.1.3	Example	306
10.2	Scenario 2: How You Embed a Web Application in an Analytic Application Through the Web Page Widget	307
10.2.1	Web Page Widget-Related postMessage and onPostMessageReceived.....	307
10.2.2	Case 1 – Posting Messages from the Host Analytic Application to the Embedded Application	307
10.2.3	Case 2 – Posting Messages from the Embedded Application to the Host Analytic Application	308
11	Scheduling a Publication.....	309
11.1	Scheduling a Publication.....	309
11.2	Working with System Configurations	311
11.3	Script API	311
11.4	Scheduling Publication Settings	312
11.5	Showing Messages for Scheduling Task.....	312
12	Data Change Insights	314
12.1	Saving a Snapshot.....	314
12.2	Listing Snapshots.....	314
12.3	Comparing Snapshots	315
12.4	Configuring Snapshot Storage.....	317
12.5	Setting Subscription Levels and Ranges	318
12.6	Setting and Getting the Version	320
13	Blend Data in Analytic Applications	322
14	Performance Best Practices	324

14.1	Top Picks	324
14.2	Analytic Application Script Performance Popup	324
14.2.1	Basic Steps	324
14.2.2	Sample Optimization Walkthrough	325
14.3	Analytic Applications	328
14.3.1	Stories and Analytic Applications.....	328
14.3.2	Browser.....	328
14.3.3	Application Design	328
14.3.4	Mobile Devices.....	328
14.3.5	Application Startup Mode.....	328
14.4	Data Sources and Widgets	329
14.4.1	Data Sources	329
14.4.2	Charts.....	329
14.4.3	Tables	329
14.4.4	Images	329
14.4.5	GeoMaps	329
14.4.6	Avoid Duplicating Widgets Unnecessarily	329
14.4.7	Consider Moving Widgets	330
14.4.8	Loading Invisible Widgets in Background	330
14.5	Scripting	332
14.5.1	Group Several setVariableValue() Calls with BW Live Connections	332
14.5.2	Prefer setDimensionFilter() Over setVariableValue() with BW Live Connections	333
14.5.3	Use getResultSet() Instead of getMembers()	333
14.5.4	Avoid Changes in onResultChanged Event Script	333
14.5.5	Avoid Modifying Data Sources or Widgets at Application Startup	333
14.5.6	Initialize Variables Via URL Parameters.....	333
14.5.7	Avoid Repeating Instructions in Loops	334
14.5.8	Prefer copyDimensionFilterFrom() over setDimensionFilter()	334
14.5.9	Enable Planning on Tables Only When Planning Is Used.....	334
14.5.10	Prefer Local Variables over Global Variables	335
14.5.11	Prefer Function Arguments over Global Variables	335
14.5.12	Use MemberInfo Object with setDimensionFilter()	335
14.5.13	Use the Pause Refresh API.....	336
14.5.14	Use Application.refreshData() to refresh multiple data sources	339
15	The End and the Future.....	340
16	Important Links	341

1 About Analytics Designer

This handbook presents the basics about SAP Analytics Cloud, analytics designer to help you understand what it's all about and how it works. Let's start with some fundamental concepts.

1.1 What Is an Analytic Application?

An **analytic application** presents data in various forms, and lets you navigate it, and enables planning. Analytic applications can range from simple static dashboards, showing static numbers, to highly customized applications. These customized applications can contain many options for browsing and navigating data, changing visualizations, and navigating across multiple pages or areas. They can have a highly customized look-and-feel, in alignment with customer branding.

1.2 What Is Analytics Designer?

Analytics designer is the functionality in SAP Analytics Cloud that allows you to create analytic applications. There is a dedicated design environment in SAP Analytics Cloud to create such applications. The term **design** doesn't refer specifically to the UX or UI design aspect of the application.

It's the entire process of creating an analytic application, which includes:

- Defining the data model
- Laying out the screen
- Configuring widgets
- Wiring it all up with the help of custom scripts

Therefore, analytics designer is another way to create analytical content in SAP Analytics Cloud.

1.3 What Can You Do with Analytic Applications That You Can't Do with Stories?

A **story** is created in a self-service workflow and can be made up of various widgets and a lot of configured functionality. However, the amount of customization is limited to the foreseen possibilities offered in the story design time environment.

An **analytic application** typically contains some custom logic, expressed with the help of scripts. With analytic applications there is much more flexibility to implement custom behavior. It requires a higher skill level to create those.

1.4 How Are Stories and Analytic Applications Related to Each Other?

In general, stories and applications share widgets and functionality to a large extent, but some widgets can only be used in applications, because they need to be scripted (dropdowns or

buttons, for example). Analytic applications can also have custom logic, which can't be implemented in stories since there is no scripting.

From a consumption point of view, there shouldn't be any difference between stories and analytic applications. The consumer shouldn't be aware of whether the analytical content is a story or an analytic application. The exposed widgets, the available functionality, and the look, feel, and behavior should be the same.

1.5 Why Do We Need Both Stories and Analytic Applications?

Stories and analytic applications share functionality and widgets and may even have very similar design environments. Why are two different artifact types necessary? The answer is that story developers and analytic application developers have completely different expectations. This is related to the differences between stories and applications:

- In the story design environment, it's practically impossible for you to create a story that doesn't work. The expectation of self-service design time for stories is that business users are guided (to some extent limited) in what they do and can do. The story design time is supposed to consist of multiple configuration steps that prevent business users from creating something which breaks. With the story design time, we ensure some level of consistency.
- It's completely different with applications, especially with the added scripts. As soon as analytic application developers add custom logic with scripting, they have complete freedom to change the default behavior of the entire analytic application. The design environment provides everything to create correct applications, but it doesn't guarantee that the application is correct or won't break.

In addition, an analytic application has a dedicated lifecycle. You start it and there are certain steps which are performed, like the startup event, for example. The story doesn't have that. You can switch the story between edit and view mode as often as you like.

These are major differences. That's why we offer two artifacts and two corresponding design time environments to create stories and analytic applications.

1.6 What Is the Typical Workflow in Creating an Analytic Application?

An analytic application is always data driven. The foundation of an analytic application is one or more underlying SAP Analytics Cloud models or a direct data access to an OData Service.

As a first step, you need to decide whether you want to visualize your data in a table or a chart and add a table or a chart to your analytic application. This triggers another step for picking a **model**. A model is a representation of the business data of an organization, organized into dimensions and measures. In addition to widgets showing data, you add to the layout other widgets that control data, such as filters, arrange and configure them, and wire them up.

Almost all widgets expose events. To add custom logic to the analytic application, you can implement event handlers with the help of the analytics designer scripting language.

1.7 What Are Typical Analytic Applications?

The variety of analytic applications is huge. analytic applications can range from very static visualizations of a few data points to very advanced, highly customized, and interactive applications which offer rich navigation and generic built-in exploration capabilities. However, there are some patterns of analytic applications:

- Table-centric data visualization

The application is comprised of a table, which consumes a large extent of the available screen real estate. Around the table, typically above it, are many user interface controls (buttons, checkboxes, dropdowns, and so on) to change the data display, such as to filter the data, change the data view, or show different dimensions. The nature of this application is that there is only one table, but many and potentially complex ways to show data differently.

- Dashboard

The application is a dashboard visualizing a few data points with the help of tiles. There is no interactivity, but it gives users an overview of highly aggregated data. A typical option of some dashboards is to use the tiles for further drilling into details: clicking on a tile takes you to a more detailed page or an entirely new application showing more details for the aggregated number on the tile.

- Generic application

Many applications are created for a specific model. That means that the user interface, the widgets, and the logic are done with knowledge of the model and its available dimensions, members, and so on. Another category is generic applications. These are applications which need to be provided with a model whenever the application is executed. These applications are more complex to create as their logic needs to work with whatever model the end user selects at runtime. The advantage is that customers don't need to create applications for each model they have maintained in their system.

1.8 How Does Scripting Work in Analytic Applications?

Almost all widgets, whether smart, data-related widgets, or simple widgets such as buttons and dropdowns, expose events. Even the analytic application itself exposes events such as a startup event or similar. To add custom logic to the application, you can implement event handlers with the help of the analytics designer scripting language.

Example:

Let's say a dropdown is populated with the available years in the data mode, for example, 2015–2019. The dropdown exposes the `onSelect` event, which is triggered whenever a value is selected from the dropdown. The implementation of that event handler could read the selected value and set a filter for the selected year in the model. The number shown reflects the selected year.

Because you can add many event handlers using the script API of all widgets and other service APIs offered, the application can be geared towards the specific needs of a customer.

1.9 What's the Scripting Language for Analytic Applications?

The scripting language is JavaScript. Scripts are executed by the web browser's JavaScript engine, which is available out of the box. To offer good tool support for analytic application developers, analytics designer adds a type system on top. This is used for the tooling and for validating scripts.

Example:

Let's say that there is a script API method available for filtering members: `setFilter("YEAR", "2014")`. A member is an element of a dimension. The plain JavaScript method expects two strings, and this is what is executed at runtime by the web browser. However, our definition of the script API method uses dedicated predefined types for our business domain, that's `setFilter(Dimension, Member)`. With that definition, the system checks and validates that the passed strings are a valid dimension and a valid member.

The script editor uses the type information. It doesn't just statically check the types but uses the knowledge about the underlying model and provides value help to offer dimensions and members existing in the model.

2 Getting Started

Analytics designer provides a software development environment that enables analytic application developers to reuse SAP Analytics Cloud widgets and other functionalities to build different kinds of applications. Interactions between different widgets, pages, and applications are implemented with script functionalities (including planning, machine learning, and so on) – at design time. End users will then be consuming these applications – at runtime.

Analytics designer is built around core story components to keep them synchronized as you go forward. Analytics designer and story have different entry points but share much in common:

- Analytics designer is deeply integrated into SAP Analytics Cloud.
- Analytics designer and story share data connectivity and User Interface artifacts.
- It ensures a consistent user experience for application and story consumers.
- It inherits infrastructure and lifecycle management of SAP Analytics Cloud.

2.1 Prerequisites

2.1.1 Required Access

Read access: the user of an analytic application needs a read access to open the application at runtime.

Full access: the application author who creates or edits the application needs a Create, Read, Update and Delete access (CRUD). The CRUD permissions are part the standard role **Application Creator** or can be assigned to any other role.

The folder where the application is stored passes on its access settings. For example, when an application is saved in a public folder, all users get Read access by default.

2.1.2 Required Roles

All standard Business Intelligence roles have a read access to consume analytic applications.

The ability to create, update, and delete is part of an extra standard role **Application Creator**.

2.1.3 Required Licenses

All SAP Analytics Cloud licenses include the creation and consumption of analytic applications. For planning applications, note the following:

- If you only need read access to existing planning models and create private versions only, you can use the **SAP Analytics Cloud for business intelligence** license.
- If you need to create public versions and use all planning features, the **SAP Analytics Cloud for planning, standard edition** is required.
- If you need to create or update a planning model for your planning application, the **SAP Analytics Cloud for planning, professional edition** license is required.

2.1.4 Modes

There are three modes in analytic applications:

- **Edit mode**

This is a design time mode. It allows you to edit applications. CRUD access is necessary. The application opens in edit mode by default if you've CRUD access.

- **Present mode**

This is a runtime mode. It allows you to execute applications. Read access is necessary. The application opens in present mode by default if you run it from edit mode.

- **View mode**

This is a runtime mode. It allows you to execute applications. Read access is necessary. The application opens in view mode by default if you've read access.

2.2 Designing Elements

For analytic applications there is a strict differentiation between design time and runtime. A few trained users create applications by using the design time elements, while many end users accessing and navigating the final application only at runtime. The following are the available designing elements.

2.2.1 Canvas

The **Canvas** is a flexible space where you can explore and present your data. Applications have only one Canvas. Scripting allows you to build numerous navigation options in your app.

2.2.2 Widgets and Filters

In Analytics designer, a **Widget** is a user interface element that can be inserted and is visible on the Canvas.

Note: Applications don't have pages. The story concepts of cascading story, page, widget filters are thus unavailable in applications. You should add a **Filter line** widget instead. The Filter line widget mimics the story filter and can be placed on the application Canvas. Assign a data-bound source widget, such as a table or a chart, as source widget. **Target** widgets can be assigned via scripting to apply the selected filters to several widgets.

For more information about widgets, see chapter [Widget Concepts, Script APIs, and Usage](#).

2.2.3 Data Sources and Models

In SAP Analytics Cloud, the widgets **table**, **chart**, and **R widget** are data-bound. They have their own data source, even if the same SAP Analytics Cloud model is connected. There is no shared data source concept. For example, you need to apply filters to each widget when you script in analytics designer for this reason.

Data Sources can be based on a variety of models provided by, for example:

- SAP HANA
- SAP BW
- Data Warehouse Cloud Analytical Data Set

2.3 Managing Your Analytic Application

2.3.1 Transporting an Analytic Application

You can import and export analytic applications from and to other SAP Analytics Cloud tenants. You can choose to export with data and other options.

Note: Custom widgets that are used in an analytic application are also exported with the analytic application.

Note: The software release wave versions of SAP Analytics Cloud installed on the source and target tenants need to be either the same wave version or just one wave version different.

2.3.2 Sharing an Analytic Application

Analytics designer has its own access. As the owner of an analytic application, you can share individual analytic applications with others and grant access to these applications.

2.3.3 Bookmarking Your Analytic Application

You, as an analytic application end user, can capture with a bookmark the current state of an analytic application after operations such as applying a filter or changing the hierarchy level.

Create Bookmark Component

To capture the state of an analytic application with a bookmark, you, as an analytic application developer, need to add a Bookmark component at design time. A bookmark version and the widgets to be included in the bookmark can be defined in the side panel of this component.



Figure 1: Bookmark Component in *Outline*

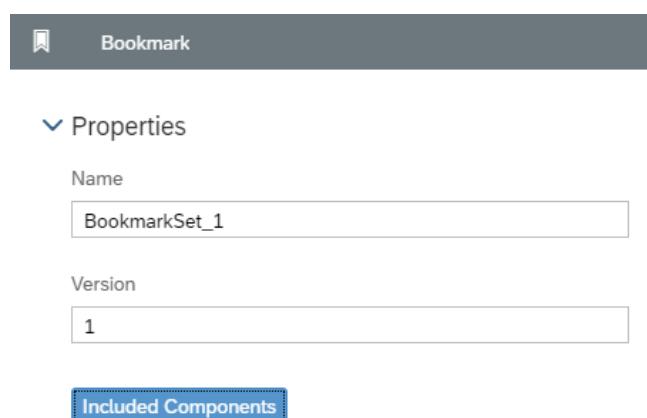


Figure 2: Side Panel of Bookmark Component

There is also an extensive Bookmark script API available. For more information, see [Bookmark API](#).

2.3.4 Translating Your Analytic Application

Translation is useful for multilingual use cases. An analytic application can be displayed in different languages in:

- The text of a widget
- The widget tooltip if applicable
- The description of the analytic application
- And so on

To turn on translation of an analytic application for the first time, the application developer must open the *Analytic Application Details* dialog and switch on *Mark for translation*.

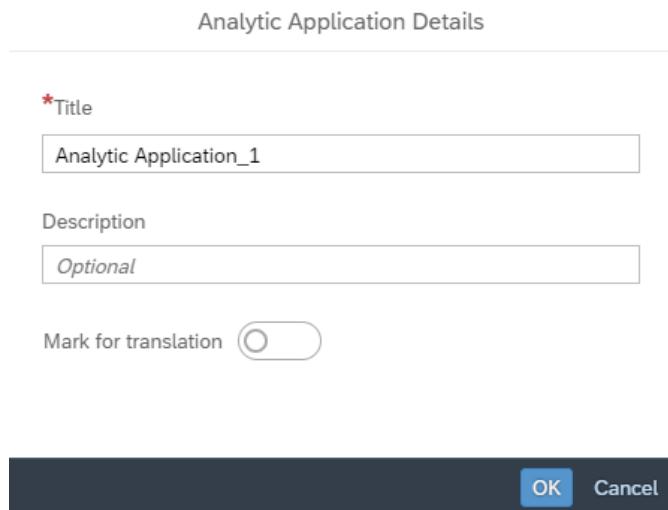
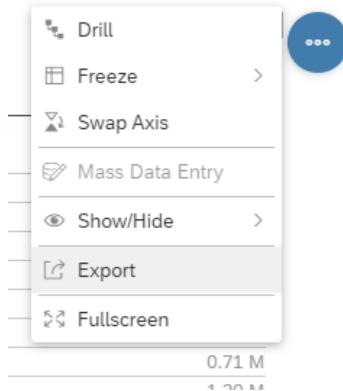


Figure 3: Turn on Translation

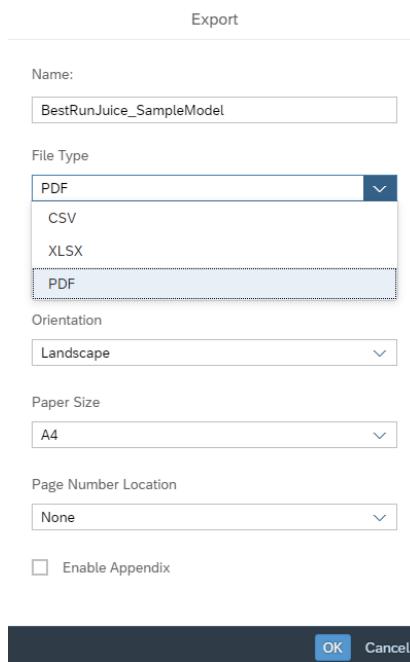
The current language will become the source language of this document. If users switch to another language, the document will be shown only in view mode.

2.3.5 Exporting Analytic Application Content

At both design time and runtime, you can export the content of a Table or a Chart widget with the widget's context menu:



To export the widget's content, select the menu item *Export*. This opens the *Export* dialog. You can choose with the *File Type* dropdown the output format (table: CSV, Excel, PDF, chart: CSV). When you choose *PDF*, you can also configure paper orientation, paper size, and so on:



When you click *OK*, the export starts.

There is also an extensive Export script API available. For more information, see [Export API](#).

3 Designing an Analytic Application

3.1 Creating an Analytic Application

To create an analytic application, you need the Application Creator role (or a custom role with the CRUD permissions) to be able to see the menu entry in the Home menu under Create.

1. Click the  menu icon,
2. click *Create*,
3. and click *Analytic Application*.

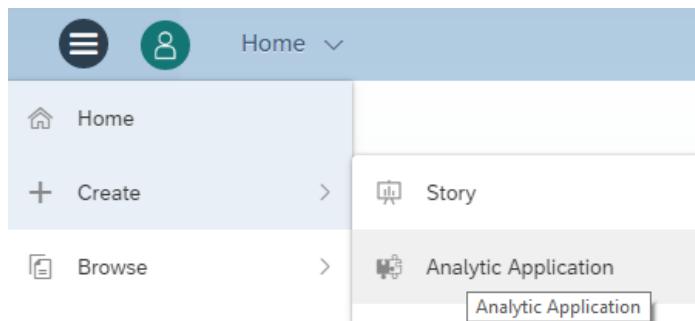


Figure 4: Create Application

3.2 Browsing for an Analytic Application

Select *Browse* under the  menu to access the file repository where are:

- Filters
- All existing public analytic applications
- Private applications
- Applications shared with you.

The default access set for an application saved in a public folder is read only for others. You need to explicitly share your application with other users and give CRUD access to allow them to edit the application.

The dialog box has sections for 'User/Team' (All Users), 'Full Access' (unchecked), 'Read Access' (checked), 'Update Access' (unchecked), and 'Delete Access' (checked). A 'Save' and 'Cancel' button are at the bottom.

Figure 5: Edit Sharing Settings

3.3 Opening Analytic Applications in a Specific Mode

For analytic applications we talk about the edit mode, where applications can be edited and the view mode, where applications are executed.

At design time, the CRUD permissions are necessary, at runtime only read access. When users have only read access and open an application from file repository, the application will open automatically in runtime mode. If a user has CRUD permissions, the application will open per default in design time mode. If you, as an analytic application developer, with CRUD permissions want to open the application from file repository directly in view mode, you can select this option from context menu when hovering over the application name in the list. If you aren't the owner of the application and it wasn't shared with full access, the application will open in view mode and you don't have the option in the context menu. Only for your own applications you've this option.

3.3.1 Opening an Analytic Application from File Repository with CRUD Permissions

If you're the owner of the application or if you've CRUD access for this analytic application, the application opens automatically **in edit mode**. The option to open the application in view mode is available in the context menu.

To open an application from a file repository in view mode:

- Hover over the application name in the list.
- Open the context menu under the  icon.
- Select *Open in view mode*.

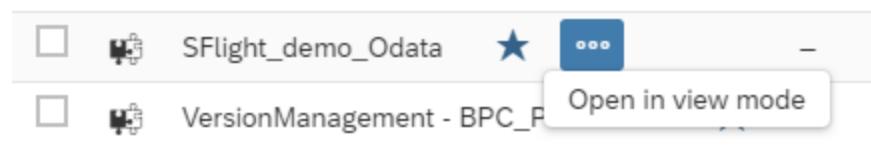


Figure 6: Open in View Mode

3.3.2 Opening an Analytic Application from File Repository with Read Permissions

If you aren't the owner of the application or if you've only read access, the application opens automatically in view mode and doesn't have a context menu entry.

3.3.3 Opening a Mode with the URL

A typical application URL looks as follows and contains a `mode`, for example:

`https://www.example.com/sap/fpa/ui/tenants/abc123/app.html#;mode=present;view_id=appBuilding;appID=xyz78`

In edit mode, the URL contains `mode=edit`. In present mode, the URL contains `mode=present`. In view mode, the URL contains `mode=view`. In embed mode, the URL contains `mode=embed`. An analytical application opened in embed mode hides the toolbar.

The analytic application opens in present mode by default when running the application from the design time.

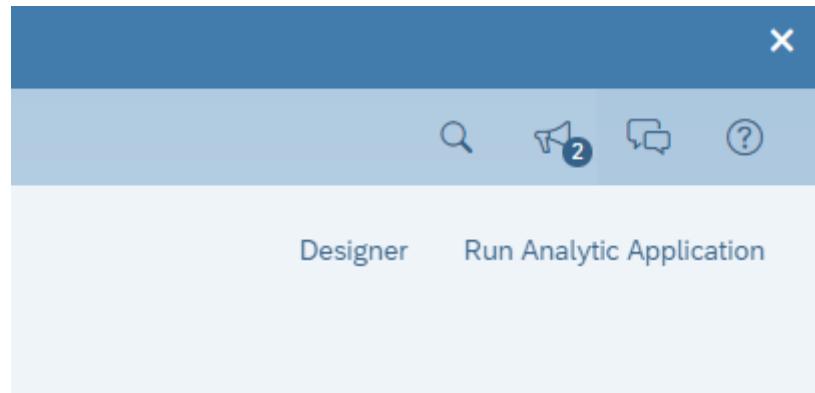


Figure 7: Run Analytic Application

To change the mode:

- Modify the URL directly or using the navigation options in the user interface.
- Click the *Fullscreen* button in the toolbar. This action changes the URL from `mode=present` to `mode=view`.

3.3.4 Switching Between Present and View Mode

You can switch between present and view mode by clicking the *Display Fullscreen* button in the toolbar. You'll notice that the URL will change. Instead of `mode=present`, now the URL contains `mode=view`.

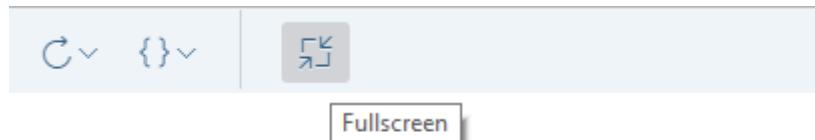


Figure 8: Fullscreen

3.4 Saving and Leaving Analytic Applications

In situations when you leave the application in edit mode while your analytic application has some unsaved changes, then the following dialog appears, which lets you save your analytic application before leaving it:

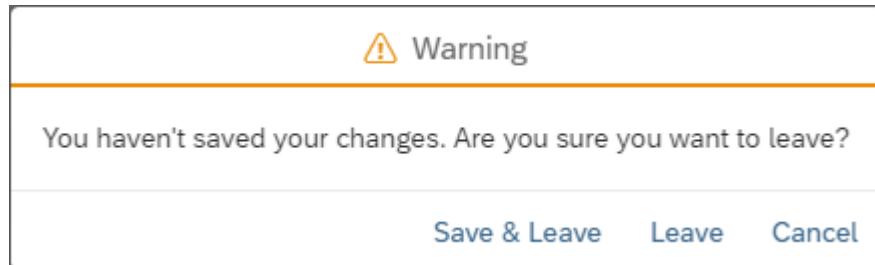


Figure 9: Save & Leave Dialog

3.5 Toolbar Functionalities

3.5.1 Toolbar in Edit Mode

As in stories there is a toolbar on top of the application which contains the features. Some options are only active once you've saved the application, otherwise they're greyed out.

- *File* contains the options like *Application Details*, *Save* and *Save As*, *Copy*, *Duplicate*, *Paste* and *Share*.
- For analytics designer you've 2 **views** which are exclusively for applications and on by default: The *Outline* and the *Info* panel. The latter contains the error list and the reference list.
- *Insert* allows you to insert Chart, Table, and all other available widgets.
- With *Tools* you can do chart scaling and create conditional formatting.
- *Data* contains refresh data and edit prompts.
- *Designer* opens the *Builder* and *Styling* panel.
- *Run Analytic Application* opens the application in another browser tab in present mode. Present mode means, that the toolbar is visible only at hover. But it can be toggled to View mode with a static toolbar by clicking on *Fullscreen* button in the toolbar.

3.5.2 Toolbar in View Mode

In view mode as well as in present mode the toolbar contains a limited set of features.

- *Data* allows you to refresh data and edit prompts.
- *Plan* contains publish data, version management, version history, value lock management, predictive forecast and allocate values.
- *Display Fullscreen* will change the mode to present mode by showing the toolbar only at hover.

3.6 Edit Mode Functionalities

3.6.1 Outline and Side Panels

The *Outline* is a crucial element of the **edit mode**. It contains:

- All visible widgets in the **Layout** area, either directly on the main **Canvas** or in a **Popup**
- The non-visible elements of an application in the **Scripting** area

Click on + to create *Script Variables*, *Script Objects*, *OData Services*, and *Predictive Models*. You can maintain them here and use them in every script of the application.

The **Outline** has a **search bar** that filters the complete tree to match your search. Click the symbol > to expand or collapse an item.

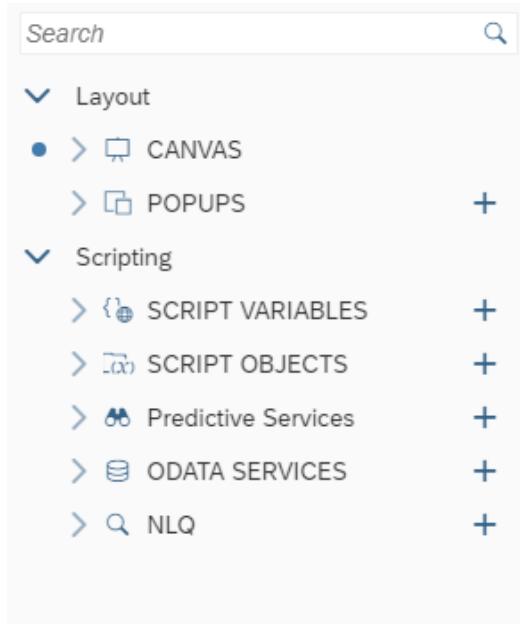


Figure 10: *Outline*

3.6.2 Scripting Section

Every **Scripting** object has a context menu that contains *Rename*, *Find Reference*, and *Delete*. When you select one of these objects, a side panel appears. It allows you to edit properties. The panel opens if you click these objects and closes when you click *Done* in the panel.

For more information, see [*Scripting in Analytics Designer*](#).

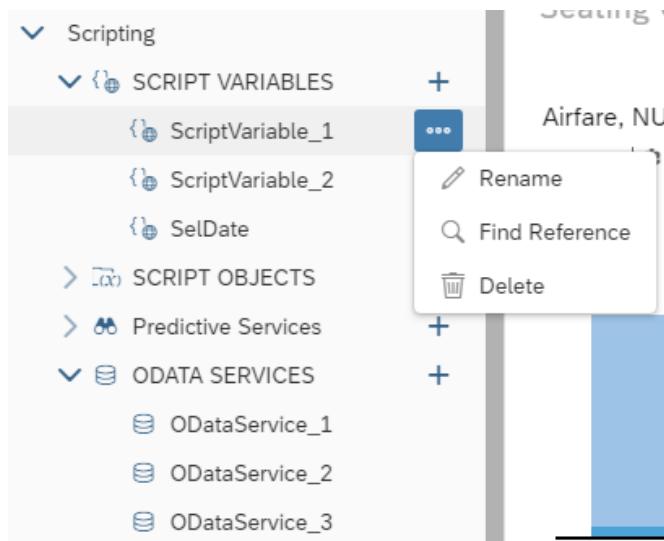


Figure 11: Context Menu for Scripting Objects in *Outline*

3.6.3 Layout Section

If the *Designer* button on the top right of the application is selected, a *Designer* panel is available for the visible widgets on the Canvas. Access the *Builder* and *Styling* panels from there.

The widgets in the *Outline*, on the Canvas, and the side panel are always synchronized and based on your selection. Widgets in the *Outline* have a context menu containing *Rename*, *Find Reference*, *Delete*, and *Hide*. *Hide* conceals the widget on the Canvas in edit mode. It has no influence on the different view modes when executing the application.

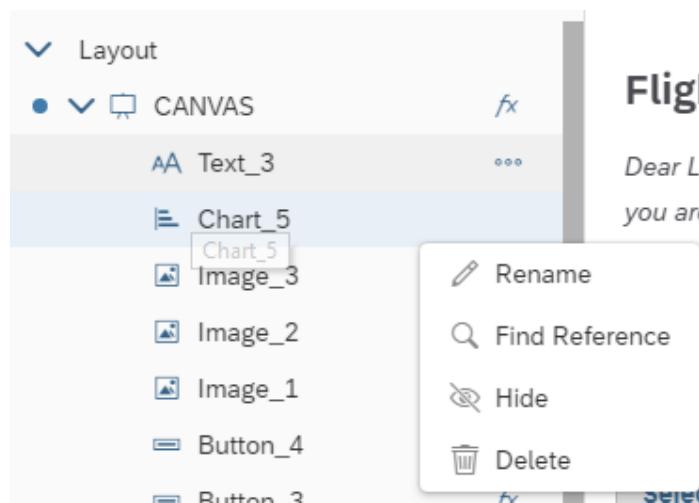


Figure 12: Context Menu for Canvas Objects in *Outline*

Widgets have their own analytic application *Properties* section in the *Styling* panel. This is where the widget name used for scripting can be changed; it's updated in the *Outline*, and vice versa. The specific properties of the analytics designer depend on the widget type.

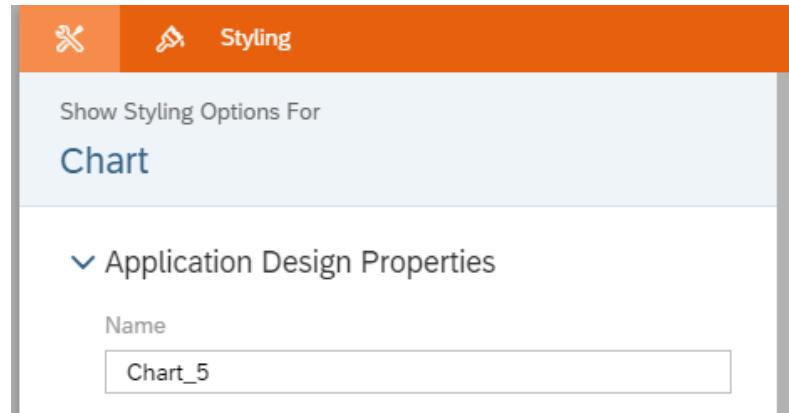


Figure 13: Widget Name

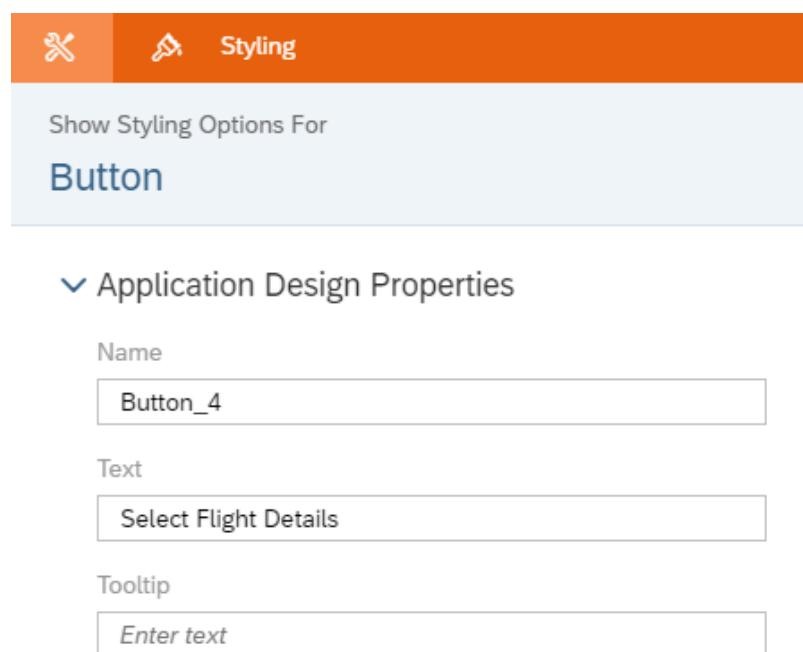


Figure 14: Analytics Designer Properties

Dropdown Widget

Users can configure dropdown styles with greater granularity. In addition to the default style, users can now configure different styles of the dropdown menu when items are selected, during mouse hover, or on mouse down.

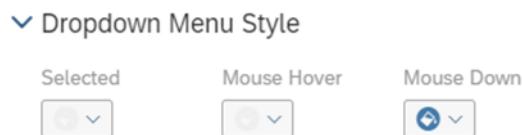


Figure 15: Dropdown Menu Style

Filter Line Widget

In addition to the default style, users can now configure different styles of filter menu during mouse hover or mouse down.

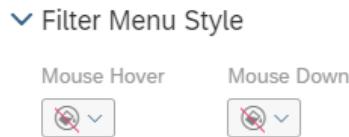


Figure 16: Filter Menu Style

Button Widget

Several new settings of Button widget have been added in the *Styling* panel:



Figure 17: Visual Feedback of Mouse Click & Hover

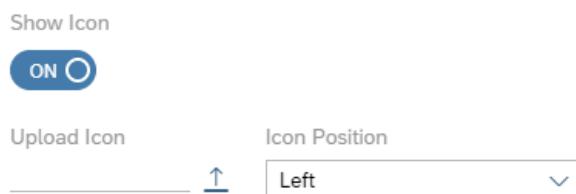


Figure 18: Settings of Icon



Figure 19: Type of Button

The possible types of buttons are the following: *standard*, *lite*, *emphasized*, *positive (accept)*, and *negative (reject)*.

Under *Actions*, you can flag the option to hide the widget in application view time.

Actions

Show this item at view time

Order



Figure 20: Actions Menu

At runtime for each widget, there are quick menus for either a widget or relevant data points (that is, Table or Chart). An application developer can configure the visibility of these quick menu items via the settings in the *Styling* panel of a widget. More styling options are available.

By checking or unchecking the checkbox before each item, the application developer can control the availability of the related quick menu item at runtime.

Note that the configurable items in *Quick Menus* vary by widget.

The screenshot shows the 'Styling' panel for a 'Chart' widget. At the top, there's a header with icons for 'Styling Options' and 'Styling'. Below the header, it says 'Show Styling Options For Chart'. A 'Close' button is on the right. Under the styling section, there's a link to 'Analytics Designer Properties'. The main area is titled 'Quick Menus' with a subtitle 'Visible in the Runtime'. A toggle switch is turned on. Below the switch is a list of items, each with a checked checkbox:

- Sorting
- Ranking
- CGR
- Fullscreen
- Export as CSV
- Create Story from Widget
- Filter/Exclude
- Drill
- Collapse/Expand
- Zoom
- Apply Axis Break
- Smart Insights

Figure 21: Quick Menu Options in *Styling* Panel

4 Scripting in Analytics Designer

4.1 Why Scripting?

You might be wondering why you would want to script and what advantage it could possibly be.

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.

4.2 Scripting Language Overview

The analytics designer scripting language is a **limited subset** of JavaScript. It's 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 hidden. Scripts are either tied to events or global script objects.

4.2.1 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's still plain JavaScript.

4.2.2 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 a script API method on an SAP BW data source, the code completion can propose measures as code completion options or values to specify a filter.

4.2.3 Events

Scripts always run in response to something happening in the application. Application events are your hook. There are several types of events in analytic applications. Some happen in the application itself and some happen on individual widgets.

4.2.3.1 Application Events

The application has two events: one that fires when the app starts, and another that's triggered in certain embedded scenarios.

- `onInitialization`: This event runs once when the application is instantiated by a user. It's where you script anything that you want to be done during startup. Like most events, it has no input parameters.

- `onPostMessageReceived`: If your application is embedded in an iFrame, your analytic application can communicate bidirectionally with the host web app using JavaScript PostMessage (see also <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>) calls. It allows the host application to pass information into the analytic application. This event is called whenever the host application makes a post message call into the analytic application.

Designers have access to this information and to the event's two input parameters:

- `origin`: It's the domain of the host application. The contents of an iFrame don't need to be in the same origin as the host app, even when same origin policies are in effect. It can be convenient but be careful about clickjacking attacks and malicious iFrame hosts. For the sake of security, we recommend checking this parameter to ensure that the iFrame host is what you expect.
- `message`: It's the standard message parameter of the JavaScript `PostMessage` passed into SAP Analytics Cloud. It doesn't follow any format and could be almost anything. It's encoded using the structured clone algorithm and there are a few documented restrictions in what can and can't be encoded.

4.2.3.2 Individual Widget Events

Most widgets have an event that's fired when the widget is clicked by a user. However, some widgets have no events, such as text labels. Data-bound widgets generally have an event that's fired when the result set of the data source changes.

Most events have no input parameters, like `onSelect` and `onResultChanged`.

4.2.4 Global Script Objects

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

4.2.5 Accessing Objects

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

4.2.6 Script Variables

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

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

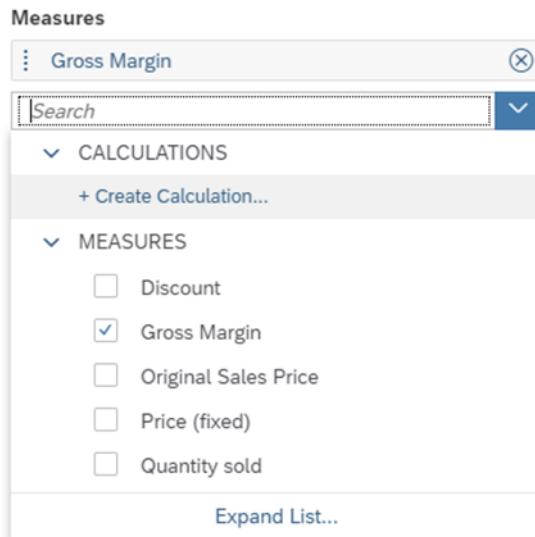


Figure 22: Create Calculation

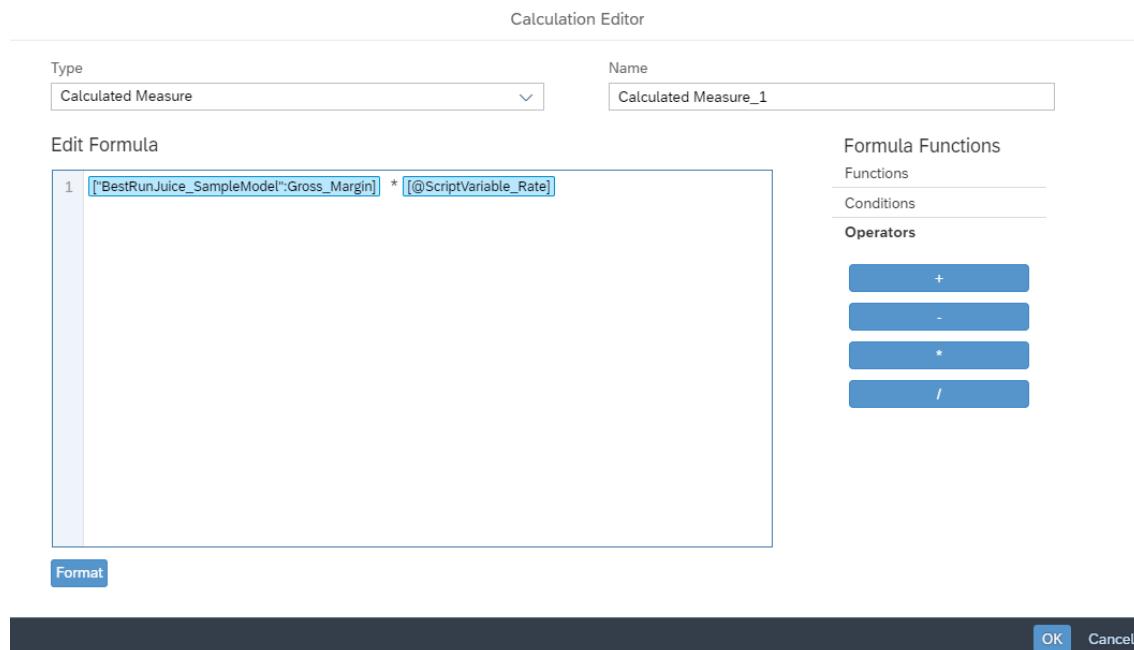


Figure 23: Reference Script Variable

4.3 Script Editor

The script editor is a tool within analytics designer to specify the actions taking place when an event is triggered by an application user. By adding a script to a widget, you can influence the behavior of this widget and thus enable user interaction, also referred to as events, at runtime. A script typically consists of several statements. A statement is a programmatic instruction within a script. The execution of a statement is typically triggered by user interaction with the widget.

4.3.1 Creating and Editing Event-Based Scripts

Scripts are presented in the *Outline*, at the left-hand side of the analytics designer editor environment.

Find them by hovering over the widget name in the *Outline*, or as a menu entry in the quick action menu of each widget. The *fx* icon indicates the event. By clicking on it, the script editor opens the selected function.

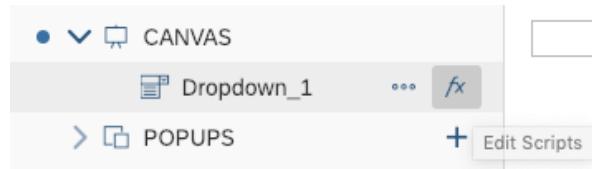


Figure 24: Edit Scripts

If a widget has multiple available events, you're presented with a choice in the hover menu.



Figure 25: Multiple Events

If there is an event with an attached script, you can see the *fx* icon in the *Outline*. If there are no attached script, there is no visible icon. In the following figure, the `onSelect` event of `Dropdown_1` has a script, but there are no scripts attached to `Chart_1`.

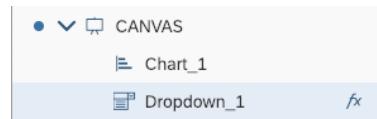


Figure 26: Script for Dropdown

If a widget has multiple events and at least one has a script attached, then the *fx* icon will be displayed.

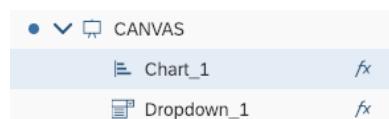


Figure 27: Script for Chart

The hover menu will show which of the events have attached scripts.

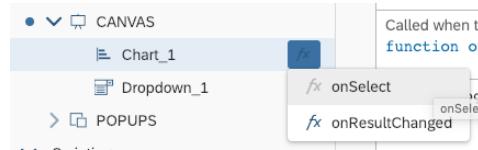


Figure 28: Hover Menu

4.3.2 Creating and Editing Functions in Global Script Objects

Functions are found under the global script objects portion of the *Outline*. Before you can add functions, you'll need to add your first script object. Do this by clicking the plus sign, next to the *Script Objects* header.

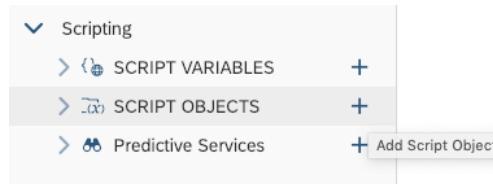


Figure 29: Add Script Object

Within a script object, you can add several functions, by invoking *Add Script Function* in the context menu. Keep in mind that the script object container is an organizational aid for you.

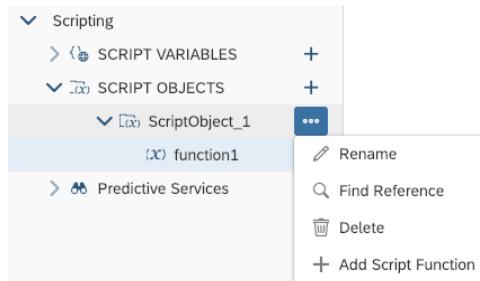


Figure 30: Add Script Function

Individual functions are nested within global script objects. For example, in the figure below you see the `function1` nested within a script object called `ScriptObject_1`.

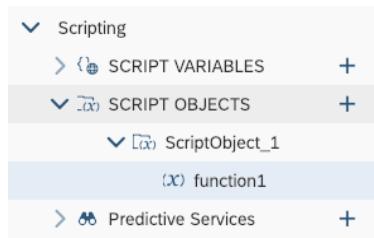


Figure 31: Script Object Function

Like Canvas widgets, the scripts attached to a function are created by clicking the icon in the hover menu of that function. Functions that have and don't have scripts are visible in the *Outline*, just as with widgets.

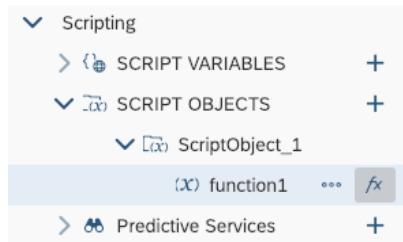


Figure 32: Script of Script Object Function

Once you've attached a script to a function, you can call it whenever you like, from any other script. You can access script objects by name. Within script objects you can access individual functions. If you wanted to invoke the `function1` script within `ScriptObject_1`, you would call it like this:

```
ScriptObject_1.function1();
```

4.3.3 Script Editor Layout

Once an open script is in the editor, it shows up as a tab along the top of the Canvas. You can open several script editor tabs at the same time.

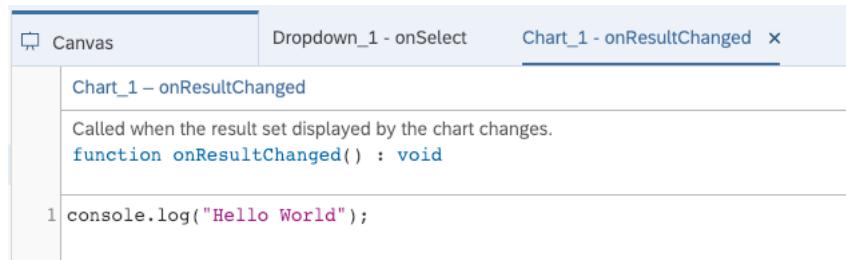


Figure 33: Script Editor

The script editor has three areas:

1. the widget and event
2. the documentation
3. the main body of the script itself

<code>Chart_1 - onResultChanged</code>	1
Called when the result set displayed by the chart changes.	
<code>function onResultChanged() : void</code> 2	
<code>1 console.log("Hello World");</code> 3	

Figure 34: Areas of Script Editor

Write script in the main body using the inbuild help features like code completion and value help.

4.3.4 Keyboard Shortcuts

The script editor provides several keyboard shortcuts, which let you, for example, undo or redo your editing operations.

Find a list of keyboard shortcuts in the help page “*Using Keyboard Shortcuts in the Script Editor*”: <https://help.sap.com/doc/00f68c2e08b941f081002fd3691d86a7/release/en-US/68dfa2fd057c4d13ad2772825e83b491.html>.

4.3.5 Info Panel: Errors and Reference List

All errors are listed in the *Errors* tab of the *Info* panel. Search for errors and filter out only warnings or errors. Double-click an error to open the script in a new tab and jump directly to the error location in the script.

Find all places where a widget or a scripting object is used with the *Find References* feature. You can find it in the context menu per object in the *Outline*. The result is displayed in the *Reference* list tab of the *Info* panel.

4.3.6 Renaming Widgets, Script Variables, and Script Functions

While creating an analytic application in analytics designer you can change the name of an analytics designer widget, component, script variable, script object, script object function, and script object function arguments. Analytics designer then applies the new name to all relevant places, for example in analytics designer scripts.

You can change the name of a widget, gadget, script variable, script object, or script object function by selecting it in the *Outline*, clicking the *More* button, selecting *Rename*, and entering a new name.

You can change the name of a widget or gadget by selecting it in the *Outline*, then entering in the *Styling* panel a new name in the *Name* input field.

You can change the name of a script variable, script object, or script object function by selecting it in the *Outline*, entering in the *Styling* panel a new name in the *Name* input field, then clicking button *Done*.

You can change the name of a script object function argument by selecting the script object function in the *Outline*, clicking the *Edit* button of the function argument in the *Styling* panel, entering a new name in the *Name* input field, then clicking button *Done*.

4.4 Scripting Language Features

4.4.1 Typing

Normal JavaScript is weakly typed and dynamically typed. Weak typing means that the script writer can implicitly coerce variables to act like different types. For example, you could have an integer value and treat it as if it were a string. 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 beforementioned integer to another type of object at will: “Dear integer, you’re now a duck”.

Analytics designer forbids both. Once you've 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's 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.

4.4.2 No Automatic Type Casting

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

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

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

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

4.4.3 Accessing Objects

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

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

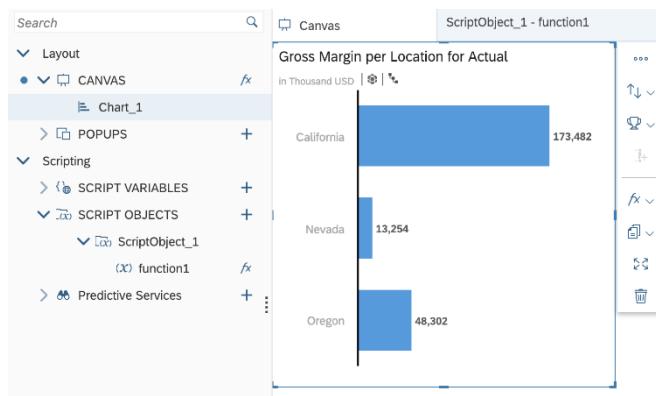


Figure 35: Accessing Objects

4.4.4 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+Spacebar`, 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've 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.

4.4.5 External Libraries

There is no provision in analytics designer for importing external JavaScript libraries. However, you can use much of the functionality of the standard JavaScript built-in objects such as:

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

The available set of standard JavaScript built-in objects and their functionality is listed in the SAP Analytics Cloud, analytics designer API Reference.

4.4.6 Debugging with console.log()

Scripts are stored as minified variables and aren't directly debuggable in the browser console. Write messages directly to the browser's JavaScript console to aid in troubleshooting. A global variable called `console` and has a `log()` function that accepts a string.

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

This would print “Hello World, 1” to the JavaScript console of the browser. Complex objects can be printed.

4.4.7 Loops

Two types of JavaScript **loops** are possible in analytics designer, `for` and `while` loops. Other types, such as `foreach` iterators, aren't supported.

4.4.7.1 for Loop

`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:

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

Instead, explicitly declare `i`. The example below is valid:

```
for (var i = 0; i < 3; i++) {
  console.log("Hello for, " + nth.toString());
}
```

4.4.7.2 while Loop

Analytics designer fully supports `while` loops:

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

4.4.7.3 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:

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

4.4.8 Double and Triple Equals Operators

Plain JavaScript has two kinds of “equals” comparison operators, `==` (double equals) and `===` (triple equals). The main difference between these is that double equals has automatic type casting while triple equals doesn’t. With triple equals, both the value and type must be the same for the result to be `true`. The triple equals is known as the strict equality comparison operator (see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness).

SAP Analytics Cloud, analytics designer has no automatic type casting. It supports

- Triple equals
- Double equals only if both sides have the same static type

The examples below show the difference between double and triple equals operators. In both cases, there is a variable `aNumber`, with an integer value and we are comparing it to the string `"1"`.

In the double equals case, `aNumber` is cast to string and compared. The result is `true`, and the if block is entered. In the triple equals case, `aNumber` isn’t cast to string and the comparison is `false`, because the values are of a different type.

This is `true`, and you can see the `if` statement is entered:

```
var aNumber = 1;
if (aNumber == "1") {
    ...
}
```

This is `false`, and you can see the `if` statement is skipped:

```
var aNumber = 1;
if (aNumber === "1") {
```

```
    ...
}
```

4.4.9 if and else Statements

The statements `if` and `else` are supported. Remember that there is no automatic type casting and double equals are valid only if both sides have the same static type:

```
if (nth === 1) {
    console.log("if...");
} else if (nth < 3) {
    console.log("else if...");
} else {
    console.log("else...");
}
```

4.4.10 this Keyword

The `this` keyword allows you to ignore the name of the object. It's simply the object that this script is attached to, regardless of what it's called. It doesn't matter and is merely a stylistic choice. With `this`, refer to

- The instance itself within widget scripts or script object functions
- The parent instance explicitly by its variable name, such as `Chart_1`
- The parent instance as `this`

When performing the above console print on one of the events of `Chart_1` itself, use the following variation of the code:

```
var theDataSource = this.getDataSource();
console.log(theDataSource.getVariables());
```

4.4.11 switch Statements

You can use normal JavaScript `switch` statements:

```
switch (i) {
    case 0:
        day = "Zero";
        break;
    case 1:
        day = "One";
        break;
    case 2:
        day = "Two";
        break;
}
```

4.4.12 break Statement

You can use `break` to break out of loops and `switch` statements, as seen in the example above.

4.4.13 Debugging Analytics Designer Scripts in the Browser

Analytics designer supports debugging analytics designer scripts with the browser's development tools.

Note: Analytics designer supports debugging analytics designer scripts in the Chrome browser only.

Note: Analytics designer transforms the analytics designer scripts before they're run in the browser. Thus, they **won't look exactly** like the script you wrote in the script editor of analytics designer.

Note: To find the analytics designer script in the browser's development tools, the script needs to be run at least once during the current session.

Analytics Designer Script Names

Analytics designer script names follow a specific naming convention: All scripts of an application are grouped in a folder <APPLICATION_NAME>. Each script is named <WIDGET_NAME>.<FUNCTION_NAME>.js.

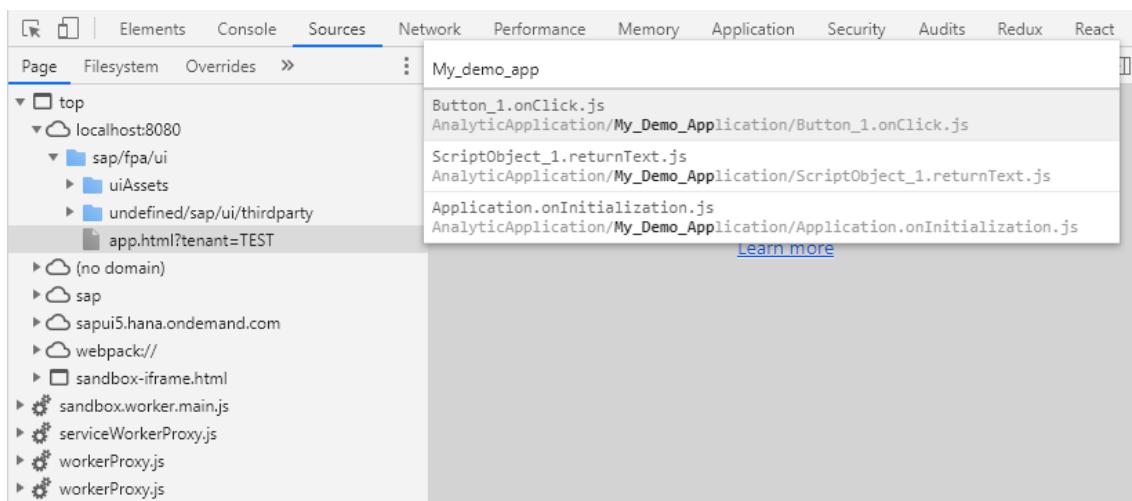
For example: If an application `My Demo Application` contains a button `Button_1` with an `onClick` event script, then the name of the script is `Button_1.onClick.js`, which is in folder `My_Demo_Application`.

Note: Special characters, for example space characters in the application name are replaced by an underscore (`_`), except minus (`-`) and dot (`.`) characters, which remain unchanged.

Find an Analytics Designer Script by Name

You can quickly find a script by its name with the following steps:

- Press `F12` to open the browser's development tools.
- Press `CTRL+P`, then start typing a part of the script's name.

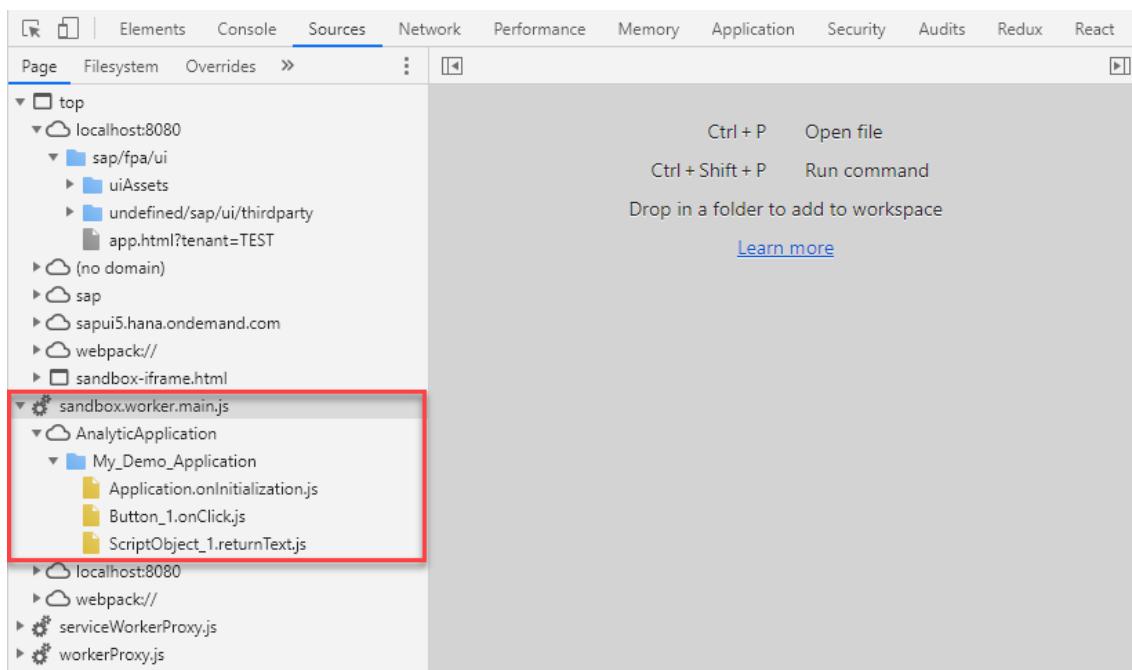


Find an Analytics Designer Script by Browsing the File Tree

You can also find a script in the file tree on the left side of the development tools using the following steps:

- Press `F12` to open the browser's development tools.

- Select the tab **Sources**.
- Open the node whose name starts with `sandbox.worker.main`.
- Open the node named `AnalyticApplication`.
- Find the folder with your application's name. The scripts that have already been executed for the current analytic application appear in this folder.

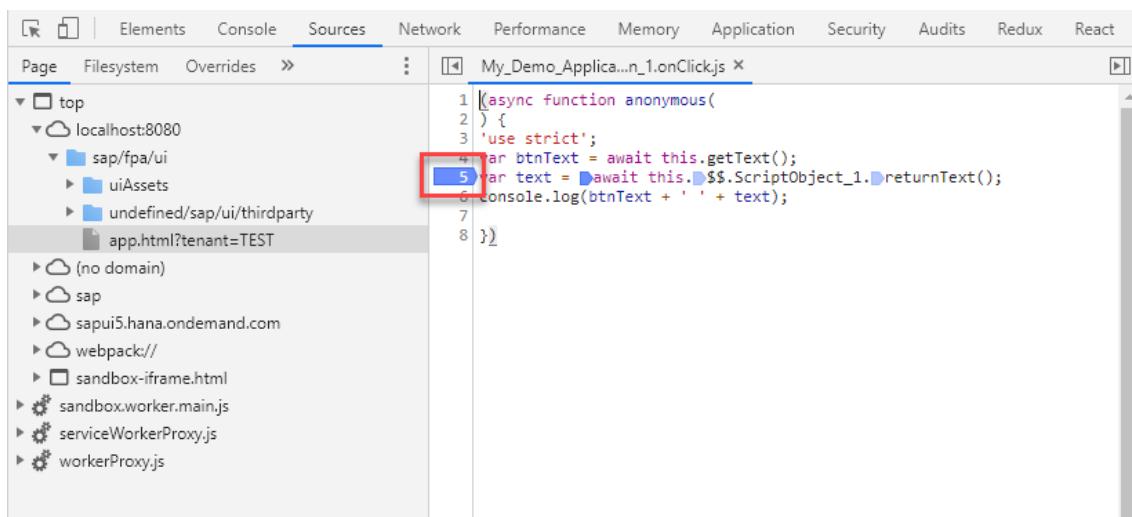


Setting and Removing Breakpoints in Analytic Designer Scripts

To pause a script while it's being executed, you can set breakpoints at certain locations in the analytic designer script.

To set a breakpoint, open the script you want to pause during its execution and click on the line number on the left side of the opened script in the developer tools.

A blue marker appears, highlighting the clicked line number. It indicates where the script will be paused when it's being executed the next time. You can add several breakpoints in one script to pause its execution at different points in time.



To remove a breakpoint, click on the blue marker. The blue marker disappears, and the script's execution won't stop at this point when the script is run the next time.

More Debugging Features

Analytic designer supports more debugging features by running an analytic application in debug mode.

You enable the debug mode for an analytic application by appending the string `;debug=true` to the URL of the analytic application in the browser.

Note: The analytic designer script names in the browser change when the analytic application is run in debug mode. In debug mode, the suffix `-dbg` is added to the script name, for example, [Button_1.onClick-dbg.js](#).

You enable the debug mode for an analytic application by appending the string `;debug=true` to the URL of the analytic application in the browser.

Note: The analytic designer script names in the browser change when the analytic application is run in debug mode. In debug mode, the suffix `-dbg` is added to the script name, for example, [Button_1.onClick-dbg.js](#).

Enabling the `debugger;` Statement

When the debug mode is enabled, you can pause an analytics designer script at a specific location while it's being executed by placing a `debugger;` statement at this location of the script. The difference to a regular breakpoint is that you can define the location where the script is paused already while writing the script itself, that's, before running it.

The screenshot shows a code editor window titled "ScriptObject_1 - returnText". The code is as follows:

```
function returnText() : string
/*
this is a block comment
*/
// debug
debugger;
// this is a comment
return Text_1.getPlainText();
```

Preserving Comments in an Analytical Designer Script

When the debug mode is enabled, then comments in a script are preserved in the transformed script that's executed in the browser. This makes it easier to recognize specifically commented locations in a script when its execution in the browser is paused.

4.4.14 Using Two-Dimensional Arrays

This section shows you how you can create and work with two-dimensional (2D) arrays in the analytics designer scripting language.

Creating 2D-Arrays

Recall that you can easily create a one-dimensional (1D) array in a script. The following snippet, for example, creates a 1D-array of numbers:

```
var arr1D = ArrayUtils.create(Type.number);
```

However, there is no similar method to create a 2D-array. Still, it's possible to create a 2D-array. The trick is to use a 1D-array that in turn contains several 1D-arrays. Together they act as a 2D-array.

Example:

In the following example, let's create a 2D-array of 4 columns and 3 rows that contains values of type `number`.

First, let's define the number of rows and columns as constants, this makes the following code clearer:

```
var numCols = 4;
var numRows = 3;
```

As mentioned before, we can't directly create a 2D-array, but we can use a trick: We create a 1D-array `arr2D` containing several 1D-arrays of type `number`. The number of 1D-arrays that `arr2D`

must contain is `numRows`, the number of rows of our array. The completed construct will have the same effect as a 2D-array.

Let's create a 1D-array that's to contain the first row of numbers:

```
var arr1D = ArrayUtils.create(Type.number);
```

Then let's create the 1D-array `arr2D` into which we plug `arr1D`, the array to contain the first row of numbers:

```
var arr2D = [arr1D];
```

At this point you may wonder why we didn't simply define `arr2D` as `var arr2D = []`? Because analytics designer won't let us, as it can't infer the content type of `arr2D` from an empty array.

Then we add the remaining 1D-arrays to `arr2D`. Each of them contains one more row of numbers. Note that the `for`-loop below doesn't start with `0` but with `1` because `arr2D` was already initialized with `arr1D`, the 1D-array to contain the first row of numbers.

```
for (var i = 1; i < numRows; i++) {
    arr2D.push(ArrayUtils.create(Type.number));
}
```

We're done!

For your reference, this is the complete code:

```
var numCols = 4;
var numRows = 3;
var arr1D = ArrayUtils.create(Type.number);
var arr2D = [arr1D];
for (var i = 1; i < numRows; i++) {
    arr2D.push(ArrayUtils.create(Type.number));
}
```

Note that we didn't need to use the constant `numCols` to define the 2D-array. However, we'll make use of it in the following sections.

Setting Values to 2D-Arrays

Now we can set a value into the array `arr2D` in the form

```
arr2D[row][col] = value;
```

For example, let's set the increasing value of a counter to each of the elements of the 2D-array `arr2D` with the following script:

```
var count = 0;
for (var row = 0; row < numRows; row++) {
    for (var col = 0; col < numCols; col++) {
        arr2D[row][col] = count;
        count = count + 1;
    }
}
```

Getting Values from 2D-Arrays

Finally, let's get and output the values from the 2D-array `arr2D` with the following script:

```
for (row = 0; row < numRows; row++) {
    for (col = 0; col < numCols; col++) {
```

```

        console.log(arr2D[row][col]);
    }
    console.log("");
}

```

The output is:

```

0
1
2
3

4
5
6
7

8
9
10
11

```

4.5 Data Sources and Working with Data

You can perform many simple operations on data. Keep in mind there are no standalone data sources, and there is a `getVariables()` function on data sources.

Example:

Let's say you want to print the variables on `Chart_1` to the console.

Get the data source of a widget with the widget's `getDataSource()` method. This returns the data source attached to that widget and allows you to perform further operations.

The snippet below prints the data source variables of `Chart_1` to the console:

```

var theDataSource = Chart_1.getDataSource();
var theVariables = theDataSource.getVariables();
console.log(theVariables);

```

4.5.1 Setting Range and Exclude Filters

The script API method `DataSource.setDimensionFilter()` supports range filters and exclude filters.

4.5.1.1 Exclude Filters

You can filter out members from the drill-down with exclude filters. The following examples show the use of exclude filters:

Example:

The following single filter value is set for dimension "`"EMPLOYEE_ID"`". This keeps only the member with employee ID 230 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {value: "230"});
```

Example:

The following single but excluding filter value is set for dimension "EMPLOYEE_ID". This removes the member with employee ID 230 from the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {value: "230", exclude: true});
```

Example:

The following multiple filter values are set for dimension "EMPLOYEE_ID". This keeps the members with employee IDs 230 and 240 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {values: ["230", "240"]});
```

Example:

The following multiple but excluding filter values are set for dimension "EMPLOYEE_ID". This removes the members with employee IDs 230 and 240 from the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {values: ["230", "240"], exclude: true});
```

4.5.1.2 Range Filters

You can filter ranges of members in the drill-down with range filters.

Note the following when using range filters:

- Range filters can only be applied to numeric dimensions.
- A time dimension isn't a numeric dimension.
- SAP BW doesn't support numeric dimensions.

The following examples show the use of range filters:

Example:

The following range filter applied to dimension "EMPLOYEE_ID" keeps the members with employee IDs between 230 and 240 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {from: "230", to: "240"});
```

Example:

The following range filter applied to dimension "EMPLOYEE_ID" keeps the members with employee IDs less than 230 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {less: "230"});
```

Example:

The following range filter applied to dimension "EMPLOYEE_ID" keeps the members with employee IDs less or equal than 230 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {lessOrEqual: "230"});
```

Example:

The following range filter applied to dimension "EMPLOYEE_ID" keeps the members with employee IDs greater or equal than 230 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {greaterOrEqual: "230"});
```

Example:

The following range filter applied to dimension "EMPLOYEE_ID" keeps the members with employee IDs greater than 230 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", {greater: "230"});
```

You can also apply multiple range filters at once.

Example:

The following range filter applied to dimension "EMPLOYEE_ID" keeps the members with employee IDs less than 230 and greater than 240 in the drill-down:

```
DS_1.setDimensionFilter("EMPLOYEE_ID", [{less: "230"}, {greater: "240"}]);
```

4.5.2 Getting Dimension Filters

You can get the dimension filters of a dimension with the method `getDimensionFilters()`. It returns an array of all filter values as `FilterValue` objects. The script API definition is as follows:

```
getDimensionFilters(string | DimensionInfo): FilterValue[]
```

Example:

In the following example, the dimension filter values of the filter `COUNTRY` are retrieved:

```
var values = Table_1.getDataSource().getDimensionFilters("COUNTRY");
```

Each value in the array is an instance of either `SingleFilterValue`, `MultipleFilterValue`, or `RangeFilterValue`, which all inherit from `FilterValue`. To work with such an instance, that is, to access its type-specific properties, cast the instance to the corresponding type first, using the `type` property. The following example shows how:

```
var value = Table_1.getDataSource().getDimensionFilters("COUNTRY")[0];
switch (value.type) {
    case FilterValueType.Single:
        var singleValue = cast(Type.SingleFilterValue, value);
        console.log(singleValue.value); // you can access the 'value' property now
        break;
    case FilterValueType.Multiple:
        var multipleValue = cast(Type.MultipleFilterValue, value);
        console.log(multipleValue.values); // you can access the 'values' property now
        break;
    case FilterValueType.Range:
        var rangeValue = cast(Type.RangeFilterValue, value);
        console.log(rangeValue.from); // you can access the 'from' property now
        console.log(rangeValue.to); // you can access the 'to' property now
        // further range properties: 'less', 'lessOrEqual', 'greater', 'greaterOrEqual'
        break;
    default:
        break;
}
```

Note: Currently this script API doesn't return time range filters.

Note: In SAP BW backend systems you can create valid filters that aren't yet supported by SAP Analytics Cloud. As this script API implementation is based on SAP Analytics Cloud, it supports the capabilities of SAP Analytics Cloud.

4.5.3 Dimension Properties

You can retrieve dimension properties of a data source.

getDimensionProperties

```
getDimensionProperties(dimension: string | DimensionInfo): DimensionPropertyInfo[]
```

Returns all available dimension properties of the specified dimension of a data source.

See also section [Dimension Properties](#) on the Dimension Properties script API of the Table.

4.5.4 Hierarchies

You can set the drill level of a dimension hierarchy as well as expand or collapse individual hierarchy nodes.

Note: Currently, this is supported only by data sources of Table and Chart widgets.

Customize the Display Level of the Hierarchy

Specify the dimension and hierarchy level that you would like to set or retrieve:

```
DataSource.setHierarchyLevel(dimension: string|DimensionInfo, level?: integer): void
DataSource.getHierarchyLevel(dimension: string|DimensionInfo): integer

// Example 1. Chart, set hierarchy level to 2
var ds = Chart_1.getDataSource();
ds.setHierarchy("Location_4nm2e04531", "State_47acc246_4m5x6u3k6s");
ds.setHierarchyLevel("Location_4nm2e04531", 2);

// Example 2. Table, set hierarchy level to 2
var ds = Table_1.getDataSource();
ds.setHierarchy("Location_4nm2e04531", "State_47acc246_4m5x6u3k6s");
ds.setHierarchyLevel("Location_4nm2e04531", 2);
```

Expand or Collapse Hierarchy Nodes

Specify the dimension and hierarchy node that you would like to expand or collapse:

```
DataSource.expandNode(dimension: string|DimensionInfo, selection: Selection): void
DataSource.collapseNode(dimension: string|DimensionInfo, selection: Selection): void

// Example 1. Chart has Location in category axis. Hierarchy info is
// "State_47acc246_4m5x6u3k6s", hierarchy level is 1.
// One node (Location="California") is expanded
Chart_1.getDataSource().expandNode("Location_4nm2e04531", {
    "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]", // California
    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Discount]"}
```

```

});

// Example 2. Chart has Location and Product in category axis.
// Hierarchy level of both location and product is 1.
// All nodes (Product="Alcohol") are expanded
Chart_1.getDataSource().expandNode("Product_3e315003an", {
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]",
  // Alcohol
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Discount]"
});

// Example 3. Table has Location and Product in row.
// Hierarchy level of both location and product is 1.
// One node (Location="California" and Product="Alcohol") is expanded
Table_1.getDataSource().expandNode("Location_4nm2e04531", {
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]",
  // Alcohol
  "Location_4nm2e04531":
  "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]", // California
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Discount]"
});

```

4.5.5 Getting Members

4.5.5.1 Getting a Single Member

You can retrieve information about a single member with the following script API:

```
DataSource.getMember(dimension: string | DimensionInfo, memberId: string,
hierarchy?: string | HierarchyInfo): MemberInfo
```

This returns the member info object of a member specified by its member ID and the member's dimension. Optionally, you can specify a hierarchy of the dimension. If no hierarchy is specified, then the current active hierarchy of the dimension is used in identifying the member.

Example:

Let's assume the current active hierarchy of dimension "[Location_4nm2e04531](#)" is set to a flat presentation, then

```
Table_1.getDataSource().getMember("Location_4nm2e04531", "CT1");
```

returns the member info object

```
{id: 'CT1', description: 'Los Angeles', dimensionId: 'Location_4nm2e04531',
displayName: 'CT1'}
```

Note: The required member ID of a member may differ, depending on the current active hierarchy of the member's dimension. For example, for a HANA system, the member ID of the same member may be "CT1" for the dimension "[Location_4nm2e04531](#)" with a flat presentation hierarchy and "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]" for an actual hierarchy. This influences the results of method `getMember()`:

Example:

Let's assume that the current active hierarchy of dimension "`Location_4nm2e04531`" is set to "`State_47acc246_4m5x6u3k6s`", a hierarchy of US-American states. Then

```
Table_1.getDataSource().getMember("Location_4nm2e04531", "CT1");
```

returns `undefined`, whereas

```
Table_1.getDataSource().getMember("Location_4nm2e04531",
  "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]");
```

returns the member info object

```
{id: '[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]', description: 'Los
  Angeles', dimensionId: 'Location_4nm2e04531', ...}
```

Example:

Let's assume that the current active hierarchy of "`Location_4nm2e04531`" is set to a flat presentation. Then

```
Table_1.getDataSource().getMember("Location_4nm2e04531", "CT1");
```

returns the member info object

```
{id: 'CT1', description: 'Los Angeles', dimensionId: 'Location_4nm2e04531', ...}
```

whereas

```
Table_1.getDataSource().getMember("Location_4nm2e04531",
  "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]");
```

returns `undefined`.

Note: If the specified hierarchy doesn't exist, then `undefined` is returned.

Note: If the data source is associated with an R visualization and you specify a hierarchy other than `Alias.FlatHierarchy` (flat presentation), then `undefined` is returned.

4.5.5.2 Getting Members

You can retrieve information about the members of a dimension with the following script API:

```
DataSource.getMembers(dimension: string | DimensionInfo, options?: integer |
  MembersOptions JSON): MemberInfo[]
```

This returns the members of the specified dimension. If you additionally specify a number, then at most this many members are returned (default: 200). If you specify members options instead of a number, then you can control the returned set of members even finer yet.

The member options are defined as

```
{
  // Type of members to retrieve (default: MemberAccessMode.MasterData)
  accessMode: MemberAccessMode
  // Hierarchy ID (default: currently active hierarchy)
  hierarchyId: string
  // Maximum number of returned members, which must be zero or a positive number.
  limit: integer
}
```

You can specify one or more member option parameters, in any sequence.

Examples:

Let's assume that the current active hierarchy of "`Location_4nm2e04531`" is set to a flat presentation. Then

```
Table_1.getDataSource().getMembers("Location_4nm2e04531", 3);
```

returns the following array of member info objects:

```
[{id: 'CT1', description: 'Los Angeles', dimensionId: 'Location_4nm2e04531', ...},  
 {id: 'CT10', description: 'Reno', dimensionId: 'Location_4nm2e04531', ...},  
 {id: 'CT11', description: 'Henderson', dimensionId: 'Location_4nm2e04531', ...}]
```

You can achieve the same result with member options, specifying the `limit` parameter:

```
Table_1.getDataSource().getMembers("Location_4nm2e04531", {limit: 3});
```

The following example retrieves the first two members and specifies a hierarchy of US-American states:

```
Table_1.getDataSource().getMembers("Location_4nm2e04531", {limit: 2, hierarchy:  
 "State_47acc246_4m5x6u3k6s"});
```

This returns the result:

```
[{id: '[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]', description:  
 'California', dimensionId: 'Location_4nm2e04531', ...},  
 {id: '[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]', description: 'Los  
 Angeles', dimensionId: 'Location_4nm2e04531', ...}]
```

In addition, you can specify the access mode, that is, whether the returned data is from master data (the default) or booked values.

The following code snippet

```
Table_1.getDataSource().getMembers("Location_4nm2e04531", {accessMode:  
 MemberAccessMode.MasterData});
```

returns the array

```
[{id: 'CT1', description: 'Los Angeles', dimensionId: 'Location_4nm2e04531', ...},  
 {id: 'CT10', description: 'Reno', dimensionId: 'Location_4nm2e04531', ...},  
 ...  
 {id: 'CT9', description: 'Las Vegas', dimensionId: 'Location_4nm2e04531', ...},  
 {id: 'SA1', description: 'California', dimensionId: 'Location_4nm2e04531', ...},  
 {id: 'SA2', description: 'Nevada', dimensionId: 'Location_4nm2e04531', ...},  
 {id: 'SA3', description: 'Oregon', dimensionId: 'Location_4nm2e04531', ...}]
```

whereas the following code snippet

```
Table_1.getDataSource().getMembers("Location_4nm2e04531", {accessMode:  
 MemberAccessMode.BookedValues});
```

returns the array

```
[{id: 'CT1', description: 'Los Angeles', dimensionId: 'Location_4nm2e04531', ...}  
 {id: 'CT10', description: 'Reno', dimensionId: 'Location_4nm2e04531', ...}  
 ...  
 {id: 'CT9', description: 'Las Vegas', dimensionId: 'Location_4nm2e04531', ...}]
```

In the second example the members of the three states California, Nevada, and Oregon aren't part of the booked values.

Tip: To find out the booked values of a dimension (in combination with a specific hierarchy), create an analytic application with a table that contains a data source with this dimension. Set the same hierarchy to the dimension. In the *Builder* panel of the table, open the ... ("three dots") menu of the dimension, then deselect the menu entry *Unbooked Values*. The table now displays the booked values of the dimension (in combination with the selected hierarchy).

Note: If the hierarchy specified in the members options doesn't exist, then an empty array is returned.

Note: If the data source is associated with an R visualization and you specify a hierarchy other than `Alias.FlatHierarchy` (flat presentation) in the members options, then an empty array is returned.

4.5.6 Getting Information About a Data Source

You can get information about a data source with the `DataSource.getInfo()` script API method. It returns a `DataSourceInfo` object containing information about the specified data source. The information is contained in a set of properties according to the following definition:

```
class DataSourceInfo {
    modelName: string,
    modelId: string,
    modelDescription: string,
    sourceName: string,
    sourceDescription: string,
    sourceLastChangedBy: string,
    sourceLastRefreshedAt: Date
}
```

Note: The `DataSourceInfo` properties `sourceName`, `sourceDescription`, `sourceLastChangedBy`, and `sourceLastRefreshedAt` are only supported for data sources based on SAP BW models. For all other models they contain the value `undefined`.

Example:

With the following sample code snippet, you can print information about a table's data source to the browser console:

```
var dsInfo = Table_1.getDataSource().getInfo();
console.log("Model name: " + dsInfo.modelName);
console.log("Model ID: " + dsInfo.modelId);
console.log("Model description: " + dsInfo.modelDescription);
console.log("Source name: " + dsInfo.sourceName);
console.log("Source description: " + dsInfo.sourceDescription);
console.log("Source last changed by: " + dsInfo.sourceLastChangedBy);
var strLastRefresh = "undefined";
if (dsInfo.sourceLastRefreshedAt !== undefined) {
    strLastRefresh = dsInfo.sourceLastRefreshedAt.toISOString();
}
console.log("Source last refreshed at: " + strLastRefresh);
```

A sample output for a data source based on an SAP BW model is:

```
Model name: HAL_TEST_Scenario_Query
Model ID: t.H:C9gjfpmu5ntxaf3dbfwtyl5wab
Model description: Sample scenario query
Source name: TEST_SCENARIO_QUERY
Source description: Test Query Scenario
Source last changed by: SYSTEM
```

```
Source last refreshed at: 2021-09-23T22:00:00.000Z
```

A sample output for a data source based on an SAP HANA model is:

```
Model name: BestRunJuice_SampleModel
Model ID: t.2.CMRCZ9NPY3VAER9A06PT80G12:CMRCZ9NPY3VAER9A06PT8314150G34
Model description: Sample Model
Source name: undefined
Source description: undefined
Source last changed by: undefined
Source last refreshed at: undefined
```

4.6 Working with Pattern-Based Functions

With pattern-based functions you can create string-based functions by only providing input and output examples instead of writing script code. For example, you can transform email addresses like `john.doe@sap.com` to names like `John Doe`.

4.6.1 Adding a Pattern-Based Function

1. In the *Outline* of your application, add a *ScriptObject*.
2. Choose ... (*More Actions*) beside the script object and select *Add Pattern Based Function*. This opens the *Pattern Based Function* dialog where you can see the generated name of the function and where you can add a description for the function.

4.6.2 Creating the Pattern

After you've added a pattern-based function to your application you need to create the pattern through input and output training examples.

1. By clicking the + symbol next to *Create Pattern* you can add an input and output example. Under *Training Example*, you can define that a certain input shall become a certain output (for example, `john.doe@sap.com` shall become `John Doe`). Note: Sometimes one example isn't enough due to potential ambiguity. In this case you can use up to three training examples by clicking the + symbol for each following example.
2. Click *Create* so that a machine learning algorithm starts looking for a pattern that matches every example. Note: As soon as you change a *Training Example*, the pattern will fall back to its default, which is just returning the input. Note: If you want to undo your changes on new or modified training examples, click *Reset*, which sets the pattern-based function to the last working pattern with the according training examples.
3. If the pattern creation has been successful, you can verify the pattern by entering input examples that follow your pattern above; otherwise, it will keep its default pattern, which is just outputting the input. To verify the pattern, click the + symbol next to *Verify Pattern*. Type a test example under *Input* to test the output: In the *Output* field you should see the correct output that follows your output pattern above.
4. When you've finished your work, click *Done*.

4.6.3 Scripting

If you've successfully created a pattern-based function and its pattern, you can use it in every script of the application just like any other script object function. The following example shows how the pattern-based function is used in a script of the application:

```
var firstNameandName = ScriptObject_1.myPatternBasedFunction("joe.doe@sap.com");
```

4.6.4 More Examples

4.6.4.1 Transforming Dates

Let's assume you've a list of dates in the form *month.day.year*, for example, *10.11.2011* and you want to transform this to *11.10.11*. So, you type *10.11.2011* as input and *11.10.11* as output in the *Training Example* section.

In this example, you want to swap the day with the month and only take the last two digits of the year. But since this is ambiguous (it's not clear if the number 11 at the end is taken from the month or from the last digits of the year) you need to provide a second training example like *09.05.2020* as an input example and *05.09.20* as an output example.

4.6.4.2 Extracting Specific Strings and Adding New Ones

Let's assume you've a list of appointments like this: *John Doe has an appointment on 06.07.20 at 3:00pm*. You want to transform this to *Name: John Doe, Date: 06.07.20, Time: 3:00pm*.

So, you type *John Doe has an appointment on 06.07.20 at 3:00pm*. as input and *Name: John Doe, Date: 06.07.20, Time: 3:00pm* as output in the *Training Example* section.

4.7 Method Chaining

Typically, scripts are written such that one line of code executes one operation. But some scripts need to be made compact, and this can be done with method chaining. Certain JavaScript libraries support method chaining where the result of a previous operation can immediately be used in the same statement. Analytics designer supports method chaining.

Suppose you were only logging the variables in the above example as a debug aid. You were not re-using them, and the multiple lines were visual clutter. Then you might want to use method chaining. The code below uses method chaining for compactness and does the same thing:

```
console.log(Chart_1.getDataSource().getVariables());
```

4.8 Script Runtime

Analytics designer validates the script before execution because running arbitrary JavaScript in the browser is a risk. It ensures that only allowed JavaScript subset can be used. Critical features like sending requests can be prevented or forced to use alternative secured APIs if needed. In addition, the execution is isolated to prevent

- Accessing the DOM
- Accessing global variables
- Modifying globals or prototypes

- Sending requests
- Importing scripts
- Including ActiveX, and so on
- Launching other Web Workers
- Accessing cookies
- Enforcing different domains

Validation

Validation at runtime follows the same logic as for the script editor. Not all validations have to be performed, for example, validating analytic data like filter values.

4.9 The R Widget and JavaScript

You might know the R widget from stories already. It becomes much more powerful in applications. The R widget has two separate runtime environments:

1. The R environment is on the server, in the R engine.
2. The JavaScript environment runs in the normal browser space along with the rest of the widget scripts.

Execution Order

On Startup, the R script runs only. The `onResultSetChanged` event script (JavaScript) doesn't run because the widget is in its initial view state.

On data change, the R script runs first, then the `onResultChanged` event script (JavaScript) runs.

Accessing the R Environment from JavaScript

The R environment can be accessed from the JavaScript environment. It can be read from and manipulated. However, the JavaScript environment can't be accessed from the R environment.

Reading

Suppose you had an R widget that had a very simple script. It just gets the correlation coefficient between two measures on a model and puts that into a number named `gmCorrelation`:

```
grossMargin <- BestRun_Advanced$`Gross Margin`  
grossMarginPlan <- BestRun_Advanced$`Gross Margin Plan`  
gmCorrelation <- cor(grossMargin, grossMarginPlan)
```

Use the `getEnvironmentValues` on the R widget to access its environment and `getNumber` to read a number from the R environment. The following JavaScript code takes the correlation coefficient from the R environment and logs it to the JavaScript console. Note the `this`. This code was taken from the `onResultChanged` event script of a widget with the above R snippet. It means that R widgets can be used as global data science scripts:

```
var nCor = this.getEnvironmentValues().getNumber("gmCorrelation");  
var sCor = nCor.toString();  
console.log("Margin Correlation: " + sCor);
```

Writing

You can also manipulate the R environment from JavaScript. The magic methods are `getInputParameters` and `setNumber`. The following line of JavaScript sets an R environment variable named `userSelection` to 0.

```
RVisualization_1.getInputParameters().setNumber("userSelection", 0);
```

4.10 Differences Between SAP Analytics Cloud and Lumira Designer

Design Studio / Lumira Designer and SAP Analytics Cloud, analytics designer have broadly similar scripting environments. Both are JavaScript based, perform similar missions and analytics designer's scripting framework was informed by experience with Design Studio. However, there are some differences that you should keep in mind.

Lumira scripts execute on the server. Analytics designer scripts execute in the browser JavaScript engine. Lumira scripts execute close to the data. Analytics designer scripts execute close to the user.

Analytics designer isn't copy-and-paste compatible with Lumira. This is partially a consequence of the close-to-data vs close-to-user philosophical difference.

Data sources are currently hidden within data-bound widgets and you must access them using `getDataSource()`. When standalone data sources become available, you can access them as global variables, as in Lumira.

Analytics designer isn't supporting automatic type conversion makes scripts more explicit and avoids common mistakes. This includes requiring a strict equality comparison operator, whereas Lumira allowed the use of the double equals comparison operator for expressions of different types.

5 Widget Concepts, Script APIs, and Usages

In analytics designer, widgets are UI elements and can be inserted onto the Canvas. There is a wide variety of widgets available. They range from basic widgets like button, text, shape, image, dropdown, checkbox group, radio button group, to data-bound ones like Table, Chart, Geo Map, and further to custom widgets built by partners and customers.

Once you've added a widget to the Canvas, you can then use its *Builder* panel, *Styling* panel, and Action Menu to configure its styling and runtime behavior, and even write script to configure how it interacts with other widgets.

If you need more information about any script API in analytics designer, you can read through the API Reference document which you can open from the help portal:

<https://help.sap.com/doc/958d4c11261f42e992e8d01a4c0dde25/latest/en-US/index.html>

5.1 Basic Widget Concepts

5.1.1 Supported Widgets

All widgets available in stories are available in analytics designer:

- Table
- Chart
- Filter Line
- Image
- Text
- Clock
- Shape
- Geo Map
- Web Page

Other widgets are available, such as:

- Dropdown
- Radio button group
- Checkbox group
- Button

Widgets can also be

- Custom-made by partners and customers

or belong to other varieties like a web page or a clock

5.1.2 Custom Widgets

You can create your own widgets with the Custom Widget SDK, which lets you extend the predefined set of widgets provided by analytics designer.

This is very useful, for example, if you need a specific user interface element, a particular visualization of data, or a certain functionality in your analytic application that isn't provided by the predefined set of widgets.

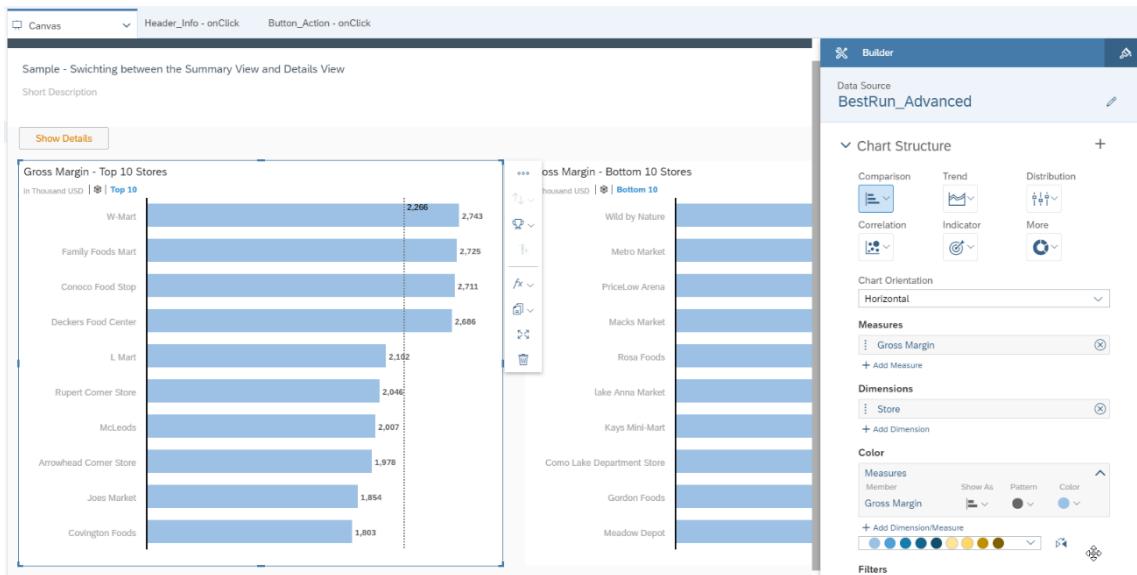
Custom widgets seamlessly integrate into analytics designer.

Find the Custom Widget SDK documentation at

<https://help.sap.com/viewer/0ac8c6754ff84605a4372468d002f2bf/latest/en-US>.

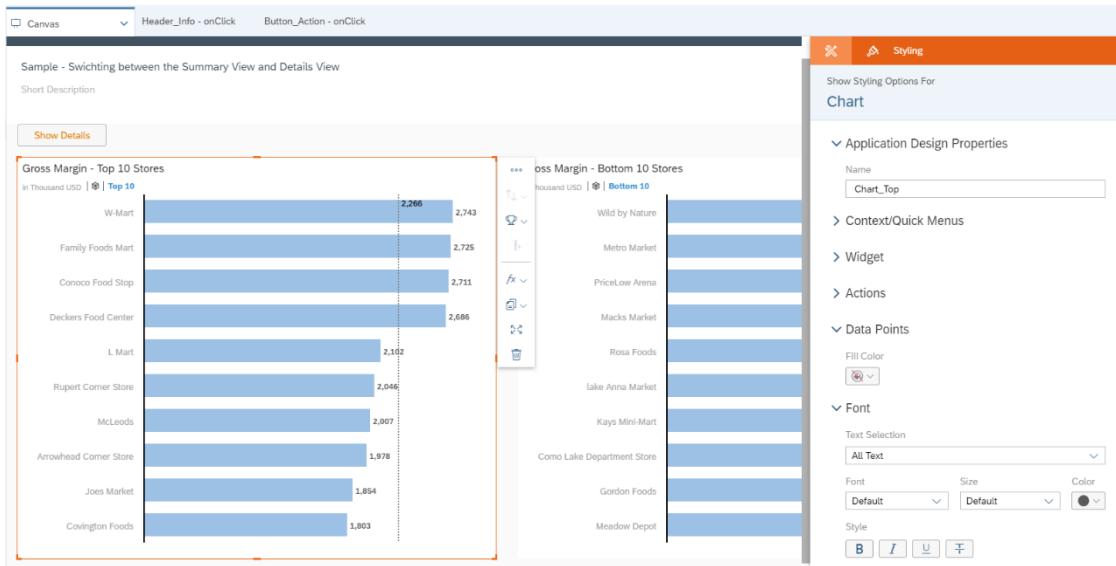
5.2 Builder Panel

If you select a widget on the Canvas, the *Builder* panel  opens on the right-hand side. With the *Builder* panel you configure your widget's data-related settings. The following example shows, how to select at least a chart type, measures, and dimension to build a chart. You can add other characteristics of this chart as well: for instance, a Reference Line. Different widgets have different configurations.



5.3 Styling Panel

You can configure the format of a widget with the help of the *Styling* panel . Multiple properties are provided with the *Styling* panel, for example background color, font, and data formats.



5.4 Action Menu

The action menu is a dynamic menu and is only visible if the widget is selected. Different widgets have different options available, and some of the options aren't available in view mode.



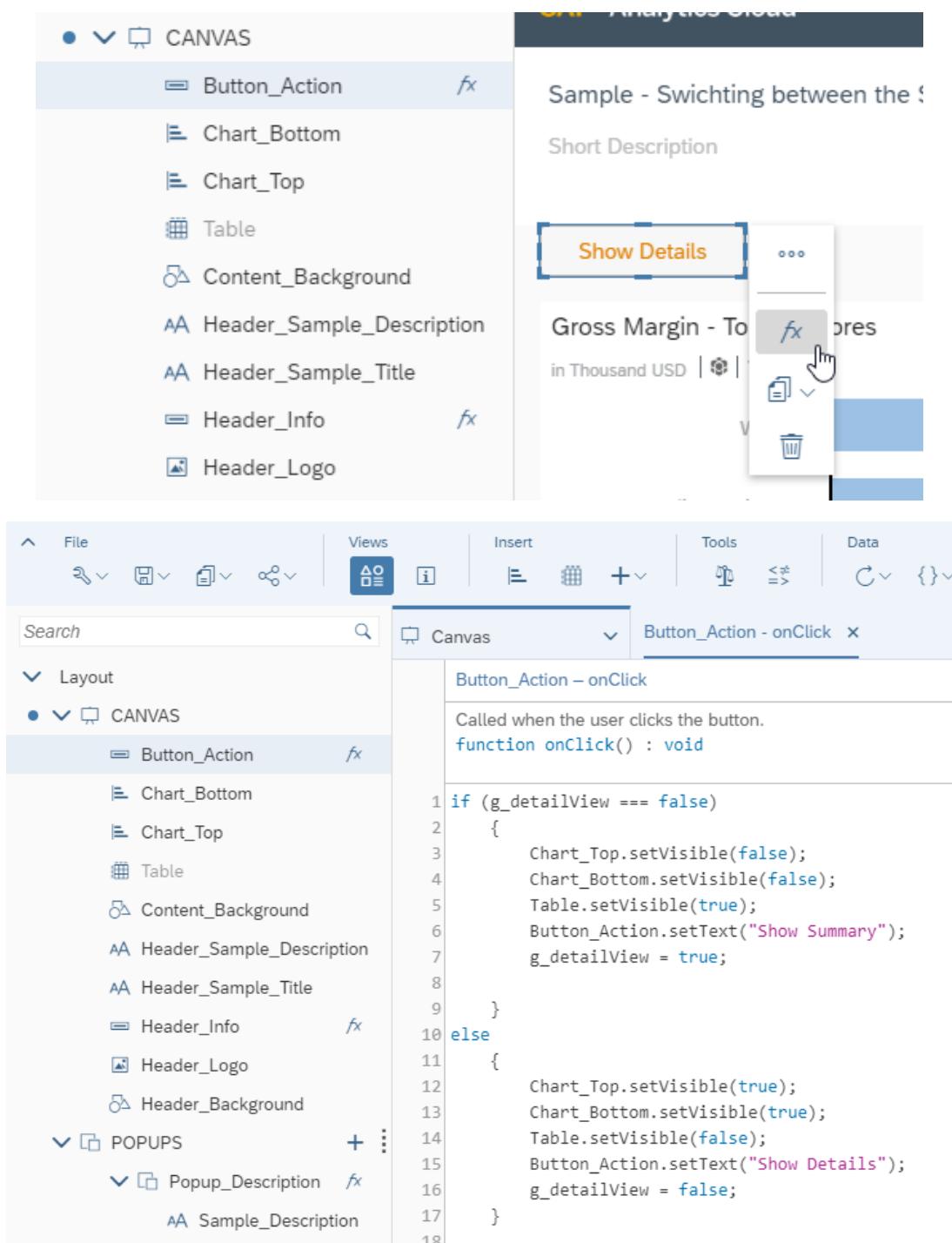
5.5 Script Editor View

Scripting provides you a powerful way to define a widget's runtime behavior and how it can interact with other widgets and other available functionality.

To edit a script function,

- Click the button in the *Action Menu*
- Or click the same button next to the widget name in the *Outline*.

It opens the *Script Editor* view.



5.6 Table

5.6.1 Table API

The Table widget displays data in rows and columns. In contrast to a chart (which is the graphical representation of data to help understand the relationship between a large quantity of data and its parts), a table is used to keep track of information such as quantities, price, text description, and other details. However, both are important means to present data and to enable end users to directly interact with data.

Analytics designer provides a Table script API and a Data Source script API to help analytic application developers add script custom specific logic into their analytic applications.

Besides the common widget script API like `getVisible()` and `setVisible()`, the main Table script API is listed below:

addDimensionToColumns

```
addDimensionToColumns(dimension: string|Dimension, position?: integer): void
```

Adds the dimension to the column axis at the specified position. If no position is specified, the dimension is added as the last dimension of the column axis.

Example:

```
Table_1.addDimensionToColumns("Location_4nm2e04531");
```

addDimensionToRows

```
addDimensionToRows(dimension: string|Dimension, position?: integer): void
```

Adds the dimension to the row axis at the specified position. If no position is specified, the dimension is added as the last dimension of the row axis.

Example:

```
Table_1.addDimensionToRows("Location_4nm2e04531");
```

getDataSource

```
getDataSource(): DataSource
```

Returns the data source of the table. If the table has no data source, `undefined` is returned. Refer to the section on the data-related script API.

getDimensionsOnColumns

```
getDimensionsOnColumns(): Dimension[]
```

Returns the dimensions on the column axis.

getDimensionsOnRows

```
getDimensionsOnRows(): Dimension[]
```

Returns the dimensions on the row axis.

getPlanning

```
getPlanning(): Planning
```

Returns the planning object of the table. If the table data source isn't of type planning, `undefined` is returned.

Refer to the section on Planning.

getSelections

```
getSelections(): Selection[]
```

Returns the selections of the chart. You can use the elements of the returned array with the function `DataSource.getData()` to get the value of a cell. For more information about the `Selection`, see <https://help.sap.com/doc/958d4c11261f42e992e8d01a4c0dde25/2019.8/en-US/doc/Selection.html>.

```
getSelections(): Selection[]
```

Returns the selections of the chart. You can use the elements of the returned array with the function `DataSource.getData()` to get the value of a cell. For more information about the `Selection`, see <https://help.sap.com/doc/958d4c11261f42e992e8d01a4c0dde25/2019.8/en-US/doc/Selection.html>.

removeDimension

```
removeDimension(dimension: string|Dimension): void
```

Removes the dimension from whichever axis it's present on. If the dimension is neither on the Rows nor Columns axis, the operation is ignored.

Example:

```
Table_1.removeDimension("Location_4nm2e04531");
```

openNavigationPanel

```
openNavigationPanel(navigationPanelOptions?: NavigationPanelOptions): void
```

Opens the navigation panel of the table. Open the navigation panel in its expanded state by specifying this in the navigation panel options:

Example:

```
Table_1.openNavigationPanel({expanded: true});
```

closeNavigationPanel

```
closeNavigationPanel(): void
```

Closes the navigation panel of the table.

Example:

```
Table_1.closeNavigationPanel();
```

5.6.2 More Table APIs

5.6.2.1 Compact Display

You can control the compact display (formerly called Universal Display Hierarchy) of a table.

Note: This script API is available on Table widgets and is applicable to SAP BW models only.

setCompactDisplayEnabled

```
setCompactDisplayEnabled(axis: TableAxis, enabled: boolean)
```

Enables or disables the compact display (formerly called Universal Display Hierarchy) on the specified table rows or columns axis.

isCompactDisplayEnabled

```
isCompactDisplayEnabled(axis: TableAxis): boolean
```

Returns whether compact display (formerly called Universal Display Hierarchy) is enabled on the specified table row or column axis.

5.6.2.2 Zero Suppression

You can control the zero suppression of values of a table.

Note: This script API is available on Table widgets and is applicable to SAP BW models only.

setZeroSuppression

```
setZeroSuppression(axis: TableAxis, enabled: boolean)
```

Enables or disables zero suppression on the specified table rows or column axis.

isZeroSuppressionEnabled

```
isZeroSuppressionEnabled(axis: TableAxis): boolean
```

Returns whether zero suppression is enabled on the specified table rows or columns axis.

5.6.2.3 Dimension Properties

You can control the active (visible) dimension properties of a table.

setActiveDimensionProperties

```
setActiveDimensionProperties(dimension: string | DimensionInfo, properties: string[] | DimensionPropertyInfo[])
```

Sets the active (visible) dimension properties of the specified dimension.

getActiveDimensionProperties

```
getActiveDimensionProperties(dimension: string | DimensionInfo): string[]
```

Returns the IDs of the currently active (visible) dimension properties of the specified dimension.

See also section [Dimension Properties](#) on the Dimension Properties script API of the Data Source.

5.6.3 Table Events

onResultChanged

```
onResultChanged()
```

Called when the result set displayed by the table changes.

onSelect()

```
onSelect()
```

Called when the user selects within the table.

5.6.4 Formatting Numbers

You can change the number formats on Table widgets at runtime, including scale unit, scale format, decimal places, and so on.

```
Table.getNumberFormat(): TableNumberFormat;
```

```
// If parameter "measures" isn't specified, format is applied to all measures
Class TableNumberFormat {
    setScaleUnit(value: NumberFormatScaleUnit, measures?: string[])
}
```

```

    setScaleFormat(value: NumberFormatScaleFormat)
    setDecimalPlaces(value: int, measures?: string[])
    setSignDisplay(value: NumberFormatSignDisplay, measures?: string[])
    setDisplayUnit(value: NumberFormatDisplayUnit)
}

NumberFormatScaleUnit.Default;
NumberFormatScaleUnit.Unformatted;
NumberFormatScaleUnit.AutoFormatted;
NumberFormatScaleUnit.Thousand;
NumberFormatScaleUnit.Million;
NumberFormatScaleUnit.Billion;
NumberFormatScaleUnit.Default;

NumberFormatScaleFormat.Long;
NumberFormatScaleFormat.Short;
NumberFormatScaleFormat.Default;

NumberFormatDisplayUnit.Default;
NumberFormatDisplayUnit.Row;
NumberFormatDisplayUnit.Column;
NumberFormatDisplayUnit.Cells;

NumberFormatSignDisplay.Default;
NumberFormatSignDisplay.MinusAsParetheses;
NumberFormatSignDisplay.MinusAsPrefix;
NumberFormatSignDisplay.PlusMinusAsPrefix;

```

5.7 Chart

5.7.1 Chart API

A chart is a graphical representation of data in symbols such as bars, lines, or slices. Analytics designer provides a Chart script API and a data source script API to help analytic application developers to add script custom specific logic into their analytic applications.

Besides the common widget script API like `getVisible()` and `setVisible()`, the main Chart script API is listed below:

addDimension

```
addDimension(dimension: string|Dimension, feed: Feed, position?: integer): void
```

Adds a dimension to the feed at the specified position. If no position is specified, the dimension is added at the end of the feed.

Example:

```
Chart_1.addDimension("Location_4nm2e04531", Feed.CategoryAxis);
```

addMeasure

```
addMeasure(measure: string|Measure, feed: Feed, position?: integer): void
```

Adds the measure to the feed, at the specified position. If no position is specified, the measure is added at the end of the feed.

Example:

```
Chart_1.addMeasure("[Account_BestRun]_sold].[parentId].&[Gross_MarginPlan]", Feed.ValueAxis);
```

getDataSource

```
getDataSource(): DataSource
```

Returns the data source of the chart. If the chart has no data source, `undefined` is returned.

Refer to the section on data-related script API

getForecast

```
getForecast(): Forecast
```

Returns the forecast of the chart.

Refer to the section on Forecast.

getMeasure

```
getMeasures(feed: Feed): Measure[]
```

Returns the measures of the feed.

Example:

```
var measures = Chart_1.getMeasures(Feed.ValueAxis);
```

getSelections

```
getSelections(): Selection[]
```

Returns the selections of the chart. You can use elements of the returned array with the function `DataSource.getData()` to get the value of a cell. For more information about the `Selection`, see <https://help.sap.com/doc/958d4c11261f42e992e8d01a4c0dde25/2019.8/en-US/doc/Selection.html>.

getSmartGrouping

```
getSmartGrouping(): SmartGrouping
```

Returns the Smart Grouping of the chart.

Refer to the section on Smart Grouping.

removeDimension

```
removeDimension(dimension: string|Dimension, feed: Feed): void
```

Removes the dimension from the feed.

Example:

```
Chart_1.removeDimension("Location_4nm2e04531", Feed.CategoryAxis);
```

removeMeasure

```
removeMeasure(measure: string|Measure, feed: Feed): void
```

Removes the measure from the feed.

Example:

```
Chart_1.removeMeasure("[Account_BestRunJ_sold].[parentId].&[GrossMarginPlan]", Feed  
.ValueAxis);
```

5.7.2 Chart Events

onResultChanged

```
onResultChanged()
```

Called when the result set displayed by the chart changes.

onSelect

```
onSelect()
```

Called when the user selects within the chart.

5.7.3 Formatting Numbers

You can change the number formats on Chart widgets at runtime, including scale unit, scale format, decimal places, and so on.

```
Chart.getNumberFormatException(): ChartNumberFormat;  
  
// If parameter "measures" isn't specified, format is applied to all measures  
Class ChartNumberFormat {  
    setScaleUnit(value: NumberFormatScaleUnit, feed: Feed)  
    setScaleFormat(value: NumberFormatScaleFormat)  
    setDecimalPlaces(value: int, measures?: string[])  
    setSignDisplay(value: NumberFormatSignDisplay, measures?: string[])  
}  
  
NumberFormatScaleUnit.Default;  
NumberFormatScaleUnit.Unformatted;  
NumberFormatScaleUnit.AutoFormatted;  
NumberFormatScaleUnit.Thousand;  
NumberFormatScaleUnit.Million;  
NumberFormatScaleUnit.Billion;  
NumberFormatScaleUnit.Default;  
  
NumberFormatScaleFormat.Long;  
NumberFormatScaleFormat.Short;  
NumberFormatScaleFormat.Default;  
  
NumberFormatSignDisplay.Default;  
NumberFormatSignDisplay.MinusAsParentheses;  
NumberFormatSignDisplay.MinusAsPrefix;  
NumberFormatSignDisplay.PlusMinusAsPrefix;
```

5.8 Result Set API

The result set-related data source script APIs allow you, as an analytic application developer, to get a result set based on an input data selection in a table or a chart, so that you can traverse it and get each data cell in the result set.

Before this script API has had been introduced, you could retrieve individual data cells using `DataSource.getData()`. However, it wasn't possible to retrieve all members for a dimension in a specific result set.

The result set-related script API mentioned above includes:

- The new script API methods `getResultSet()` and `getDataSelections()` – exposed on the data source of a Chart or Table widget to fetch all data cells and iterate over its result.
- The new script API method `getResultMember()` – exposed on the data source of a Chart or Table widget to fetch member specific information.

Note: To reference in a selection the NULL member, use the alias `Alias.NullMember`.

Note: To reference in a selection the totals member, use the alias `Alias.TotalsMember`.

5.8.1 Using the `getResultSet` API

The `getResultSet()` script API method is exposed on the data source of both the Chart and Table widget. In this section, we will take charts and tables as example and show you how to fetch data cells from such widgets. Users can specify input parameters to filter the result. If there is no input parameter passed to this script API method, all data cells are returned. When a table has newly added cells at runtime, these cells are also returned by this script API method.

To help you understand using this script API method, we list several examples. As ID of dimension and measure is used in input parameter and returned by the result set, we choose to display both ID and description for tables and charts in these examples.

Function Summary:

```
// Returns the result set according to the selected data or context of
// the data you select. Offset / limit should be not less than zero. If
// offset / limit are invalid or not set, all data is returned.
// If the selection doesn't specify any MeasureDimension, all measures
// are returned.

Chart_1.dataSource().getResultSet(selection?: Selection | Selection[] | SelectionContext, offset?: integer, limit?: integer): ResultSet[]
Table_1.dataSource().getResultSet(selection?: Selection | Selection[] | SelectionContext, offset?: integer, limit?: integer): ResultSet[]

ResultSet {
    [key: string]: DataContext;
}

Selection {
    [key: string]: string;
}

SelectionContext {
```

```

    [key: string]: [string];
}

DataContext {
  id: string
  description: string
  formattedValue: string
  rawValue: string
  parentId: string
  properties?: { // "properties" doesn't contain "id" or "description"
    [key: string]: [string]
  }
}

```

Example 1: Get a result set when no input parameter is specified

Here is an example that shows the result when no input parameter is specified: All data points of [Chart_1](#) are returned. Dimension context includes parent information if it has a hierarchy structure. Measure context includes `formattedValue` and `rawValue`.

The below chart shows *Gross Margin per Location*. *Include Parent Levels* has been checked in the *Builder* panel.

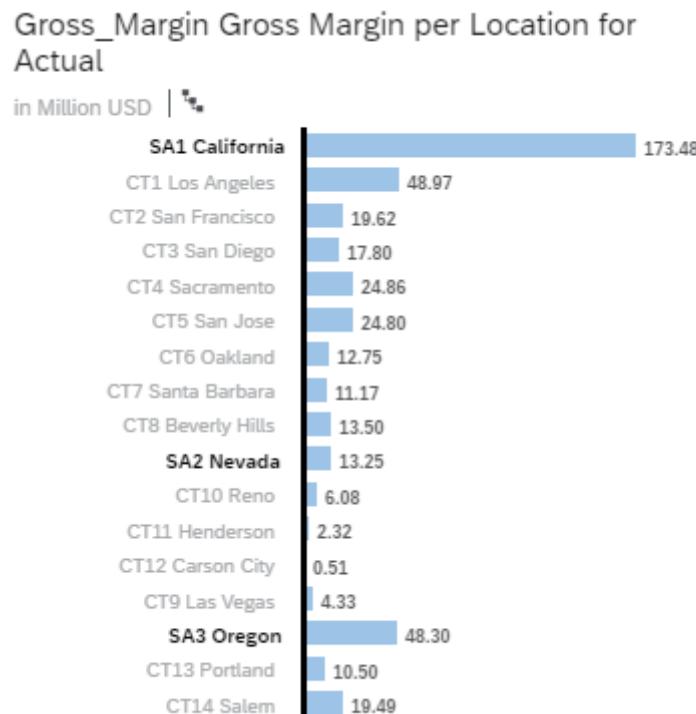


Figure 36 Gross Margin per Location Chart

```

Chart_1.getDataSource().getResultSet();

// ResultSet[].
// Both CT1 and CT2 have parentId "SA1". The result set is partially listed.
[{
  "@MeasureDimension": {

```

```

    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "formattedValue": "173.48",
    "rawValue": "173481592.97"
  },
  "Location_4nm2e04531": {
    "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
    "description": "California"
  }
}, {
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "formattedValue": "48.97",
    "rawValue": "48971999.74"
  },
  "Location_4nm2e04531": {
    "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
    "description": "Los Angeles",
    "parentId": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]" // CT1's
parentId is "SA1"
  }
}, {
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "formattedValue": "19.62",
    "rawValue": "19619690.4"
  },
  "Location_4nm2e04531": {
    "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT2]",
    "description": "San Francisco",
    "parentId": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]" // CT2's
parentId is "SA1"
  }
}, {
  ...
}]

```

Example 2: Get a result set when a data point in a chart has more than one measure

If a chart has more than one measure, such as a bubble or scatter chart, each measure combined with its dimension context represents a cell in the `ResultSet` array. For example, one data point of a bubble chart has three cells in the `ResultSet` array. They represent *Gross Margin Per Location*, *Profit Per Location*, and *Original Sales Price Per Location*, respectively.

```

Chart_1.getDataSource().getResultSet();

// ResultSet[].
// Three cells of ResultSet array represent one data point of a bubble chart.
[{

```

```

"@MeasureDimension": { // Gross_Margin on X axis
  "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
  "description": "Gross Margin",
  "formattedValue": "48.97",
  "rawValue": "48971999.74"
},
"Location_4nm2e04531": {
  "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
  "description": "California"
},
{
  "@MeasureDimension": { // Profit on Y axis
    "id": "[Account_BestRunJ_sold].[parentId].&[Profit]",
    "description": "Profit",
    "formattedValue": "19.62",
    "rawValue": "19619690.4"
},
"Location_4nm2e04531": {
  "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
  "description": "California"
},
{
  "@MeasureDimension": { // Original_Sales_Price as bubble size
    "id": "[Account_BestRunJ_sold].[parentId].&[Original_Sales_Price]",
    "description": "Original Sales Price",
    "formattedValue": "0.82",
    "rawValue": "819975.23"
},
"Location_4nm2e04531": {
  "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
  "description": "California"
}
},
...
}]

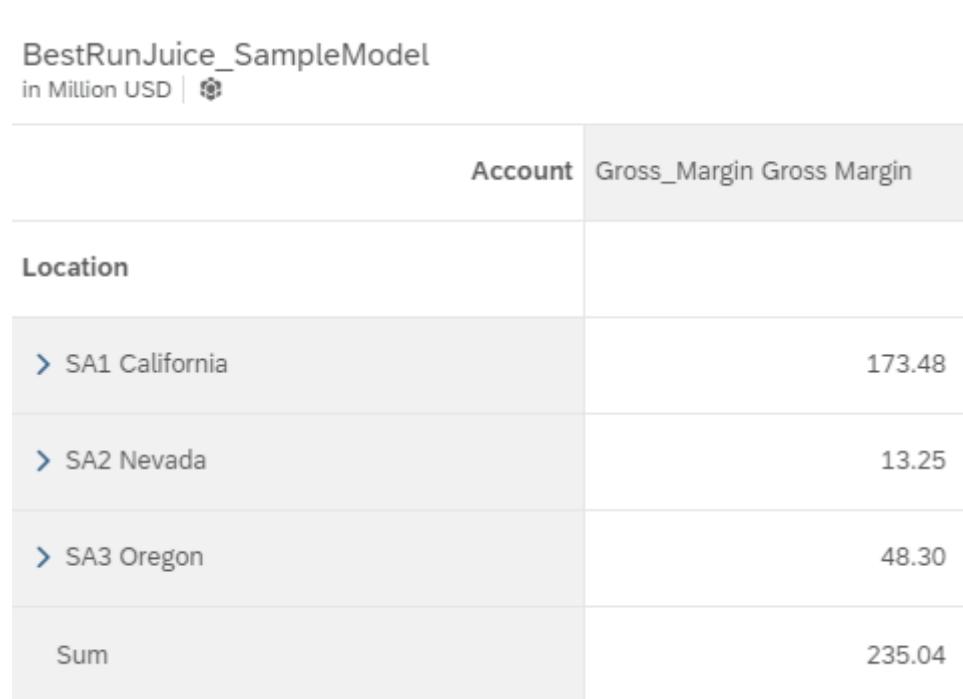
```

5.8.1.1 Adding New Cells to a Table

Using the script API for a table is the same as for a chart. Below are some specific features when using the script API for a table that affect the results returned by the `getResultSet()` script API method if they're enabled.

At both design time and runtime, additional columns can be added to the table by right-clicking a table cell and adding a calculation. As these generated columns are part of the table, they're also returned in the result set.

Example 3: Newly added row has the description "Sum" and its ID is a UUID



Location	Account	Gross Margin
> SA1 California		173.48
> SA2 Nevada		13.25
> SA3 Oregon		48.30
Sum		235.04

Figure 37 Newly Added Row Has Description "Sum", ID Is a UUID

```
Table_1.getDataSource().getResultSet();

// ResultSet[]. Additional row is generated in table.
[{
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "formattedValue": " 235.04",
    "rawValue": " 235036949.37"
  },
  "Location_4nm2e04531": {
    "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
    "description": "California"
  }
}, {
  ...
}, {
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "formattedValue": "48.30",
    "rawValue": "48301721.3"
  },
  "Location_4nm2e04531": {
    "id": "30835424-2216-4348-9273-946191519134", // generated in frontend
    "description": "Sum"
  }
}]
```

```
}]
```

Example 4: Totals member is generated in backend by selecting “Show Totals” in Builder Panel

In this case, the ID of a totals member is an alias (@TotalMember).

BestRunJuice_SampleModel in Million USD 		
	Account	Gross Margin
Location		
Totals		235.04
➤ SA1 California		173.48
➤ SA2 Nevada		13.25
➤ SA3 Oregon		48.30

Figure 38 Totals Member Is Generated by Choosing *Show Totals*

```
Table_1.getDataSource().getResultSet();

// ResultSet. Totals is generated in backend.
[{
  "@MeasureDimension": {
    "id": "[Account_BestRun]_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "formattedValue": "235.04",
    "rawValue": "235036949.37"
  },
  "Location_4nm2e04531": {
    "id": "@TotalMember", // id of "Totals", it's an alias
    "description": "Totals"
  }
}, {
  ...
}]
```

Example 5: Comment row or column is added to table at frontend

In the below example, a comment column is added to a table and one comment cell is input with text.

Version	Actual	Comment
Account		
▼ TOTAL	61,254,901.96	comment test
▼ EBIT	61,254,901.96	
▼ Gross Profit	61,254,901.96	
Revenue	28,960,784.31	
COGS	32,294,117.65	

Figure 39 Comment Row or Comment Column Is Added to Table in Frontend

```

Table_1.getDataSource().getResultSet();

// ResultSet[]
[{
  "Version": {
    "id": "public.Actual",
    "description": " Actual"
  },
  "@MeasureDimension": {
    "id": "[Account].[parentId].&[TOTAL]",
    "description": "TOTAL",
    "rawValue": "61254901.96",
    "formattedValue": "61,254,901.96"
  }
}, {
  "Version": {
    "id": "24769565-5194-4582-9346-349222998310",
    "description": " Comment"
  },
  "@MeasureDimension": {
    "id": "[Account].[parentId].&[TOTAL]",
    "description": "TOTAL",
    "rawValue": "comment test",
    "formattedValue": "comment test"
  }
}, {

```

```
...
}]
```

The table row can be hidden, removed, or excluded. In these cases, only visible rows are returned by the `getResultSet()` script API method.

If there are many rows in one table and a scroll bar is displayed, even though some of the rows are temporarily invisible, the `getResultSet()` script API method still returns all data cells in this table because all data is already fetched and sent to the frontend.

5.8.1.2 Specifying Input Parameter and Filter Result

As the result set of a chart or a table may have many cells, users can filter the result set by conditions. The `getResultSet()` script API method accepts a `Selection`, a `Selection` array, and a `SelectionContext` as an input parameter.

Users can specify a concrete selection of a data cell to get its `description`, `parentId`, `formattedValue`, and `rawValue`.

Example 6: Specify input parameter, location equals CT1

```
// Input parameter is a concrete Selection, only one cell is returned
Chart_1.dataSource().getResultSet({
    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]"
});

// ResultSet[]. One data cell is returned.
[{
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
        "description": "Gross Margin",
        "formattedValue": "48.97",
        "rawValue": "48971999.74"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
        "description": "Los Angeles",
        "parentId": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]"
    }
}]
```

Example 7: Specify Selection parameter, location dimension member equals CT1 or CT2

The `Selection` parameter can be an array and multiple filters are supported. The below condition describes that the location dimension member should equal CT1 or CT2.

```
// Input parameter is Selection[]
Chart_1.dataSource().getResultSet([
    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "Location_4nm2e04531": [
        "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]"
    ],
    {
        "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
        "Location_4nm2e04531": [
            "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT2]"
        ]
    }
})
```

```

    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "Location_4nm2e04531":
    "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT2]"
});

// ResultSet[]. Two data cells are returned.
[{
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
        "description": "Gross Margin",
        "formattedValue": "48.97",
        "rawValue": "48971999.74"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
        "description": " Los Angeles",
        "parentId": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]"
    }
}, {
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
        "description": "Gross Margin",
        "formattedValue": "19.62",
        "rawValue": "19619690.4"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT2]",
        "description": "San Francisco",
        "parentId": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]"
    }
}]

```

Example 8: Specify input parameter via SelectionContext

We also introduce the [SelectionContext](#) as a parameter. As measure is the same (*Gross Margin*) in this case, the measure ID is specified only once. The below example has the same result set as the previous one.

```

Chart_1.getDataSource().getResultSet({
    "@MeasureDimension": "[[Account_BestRunJ_sold].[parentId].&[Gross_Margin]]",
    "Location_4nm2e04531":
    "[[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]]",
    "[[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT2]]"
});

```

Example 9: Selection specified by input parameter isn't targeted to a certain data cell

If the input parameter isn't a concrete data cell selection, all data cells matching this condition are returned.

BestRunJuice_SampleModel in Million USD		Account	Gross_Margin	Gross Margin	Discount	Discount
Location	Product					
> SA1 California	> PC4 Alcohol		25.39		30.72	
	> PC1 Carbonated Drinks		44.26		15.65	
	> PC2 Juices		102.78		130.30	
	> PC3 Others		1.05		1.06	
> SA2 Nevada	> PC4 Alcohol		1.53		6.54	
	> PC1 Carbonated Drinks		4.62		3.77	
	> PC2 Juices		7.04		32.11	
	> PC3 Others		0.06		0.21	
> SA3 Oregon	> PC4 Alcohol		7.11		15.05	
	> PC1 Carbonated Drinks		13.20		7.48	
	> PC2 Juices		27.68		59.71	
	> PC3 Others		0.31		0.53	

Figure 40 Input Parameter Isn't a Concrete Data Cell Selection

```
// Input parameter is Selection. Product member isn't specified.

Table_1.getDataSource().getResultSet({
    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "Location_4nm2e04531":
    "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]"
});

// ResultSet[]. Four data cells are returned.
[{
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
        "description": "Gross Margin",
        "rawValue": "25388481.78",
        "formattedValue": "25.39"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
        "description": "California"
    },
    "Product_3e315003an": {
        "id": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]",
        "description": "Alcohol"
    }
}, {
    ...
}]
```

Example 10: Input parameter selections overlap one another

If selections overlap, they're treated as an OR operation.

```
// Input parameter is Selection[] and result of one selection overlap
// with another.

Table_1.getDataSource().getResultSet([
    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "Location_4nm2e04531":
    "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
    "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]"
    // one data cell
], {
    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Discount]",
    "Location_4nm2e04531":
    "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA2]",
    "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC1]"
    // one data cell
}, {
    "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]"
    // four data cells
}]);

// ResultSet[].
// Five data cells are returned as one data cell is selected by two selection.

[{
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
        "description": "Gross Margin",
        "rawValue": "25388481.78",
        "formattedValue": "25.39"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
        "description": "California"
    },
    "Product_3e315003an": {
        "id": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]",
        "description": "Alcohol"
    }
}, {
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Discount]",
        "description": "Discount",
        "rawValue": "3765388.14",
        "formattedValue": "3.77"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA2]",
        "description": "Nevada"
    },
    "Product_3e315003an": {
        "id": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC1]",
```

```

        "description": " Carbonated Drinks "
    }
},
...
}]

```

Example 11: How to define and use parameter offset and limit

Parameter offset and limit are used to limit the amount of data cells.

Offset skips the offset cells before beginning to return the cells.

Limit constrains the number of returned cells.

```

// Only two data cells are returned.
Table_1.getDataSource().getResultSet(null, 0, 2);

// ResultSet[]. Two data cells are returned.
[{
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
        "description": "Gross Margin",
        "rawValue": "25388481.78",
        "formattedValue" "25.39"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
        "description": "California"
    },
    "Product_3e315003an": {
        "id": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]",
        "description": "Alcohol"
    }
}, {
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Discount]",
        "description": "Discount",
        "rawValue": "30720120.73",
        "formattedValue" "30.72"
    },
    "Location_4nm2e04531": {
        "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
        "description": "California"
    },
    "Product_3e315003an": {
        "id": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]",
        "description": "Alcohol"
    }
}]

```

5.8.1.3 Using Forecast and Smart Grouping

If the forecast feature is enabled in a Time Series chart, the `getResultSet()` script API method returns both the actual value and the forecast value.

The data point context of the actual date has the measure `actualValue`. If there is fitted line in this date, another measure `hindcastValue` can be found.

The data point context of the forecast date has three measures: `upperBound`, `lowerBound`, and `forecastValue`.

Example 12: Time Series chart with forecast enabled, extra data point is generated

The Time Series chart below has both an actual date and a forecast date. The blue area is generated by the forecast.

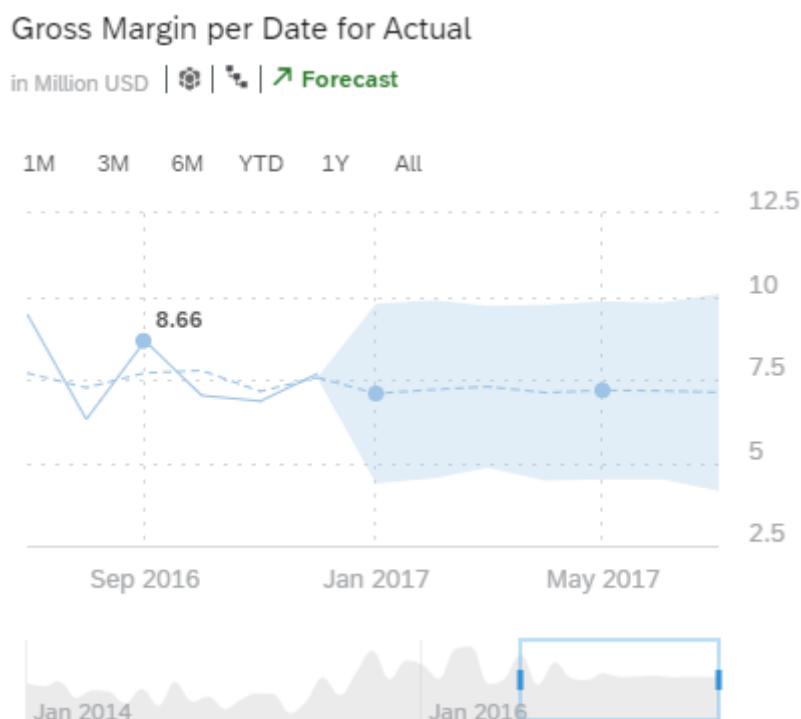


Figure 41 Time Series Chart with Forecast Enabled

```
Chart_1.getDataSource().getResultSet();

// ResultSet[]
[{
  "Date_703i1904sd.CALMONTH": {
    "id": "201409",
    "description": "Sep 2014" // actual date
  },
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "rawValue": "4207974.43",
    "formattedValue": "4207974.43"
  }
}]
```

```
},
"@Forecast": {
  "id": "actualValue"
}
}, {
  "Date_703i1904sd.CALMONTH": {
    "id": "201409",
    "description": "Sep 2014" // actual date
  },
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "rawValue": "5903071.626874865",
    "formattedValue": "5903071.626874865"
  },
  "@Forecast": {
    "id": "hindcastValue" // fitted line in dashed line
  }
},
...
{
  "Date_703i1904sd.CALMONTH": {
    "id": "1498867200000",
    "description": "Jul 2017" // forecast date
  },
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "rawValue": "7115040.662185087",
    "formattedValue": "7115040.662185087"
  },
  "@Forecast": {
    "id": "forecastValue"
  }
},
{
  "Date_703i1904sd.CALMONTH": {
    "id": "1498867200000",
    "description": "Jul 2017" // forecast date
  },
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
    "description": "Gross Margin",
    "rawValue": "10076952.088587102",
    "formattedValue": "10076952.088587102"
  },
  "@Forecast": {
    "id": "upperBound"
  }
}, {
```

```

    "Date_703i1904sd.CALMONTH": {
      "id": "149886720000",
      "description": "Jul 2017" // forecast date
    },
    "@MeasureDimension": {
      "id": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
      "description": "Gross Margin",
      "rawValue": "4153129.2357830727",
      "formattedValue": "4153129.2357830727"
    },
    "@Forecast": {
      "id": "lowerBound"
    }
  ]
}
  
```

Example 13: Bubble chart with smart grouping enabled, extra measure is generated

If smart grouping is enabled in a bubble chart, it generates one extra measure to identify the group of each data point.

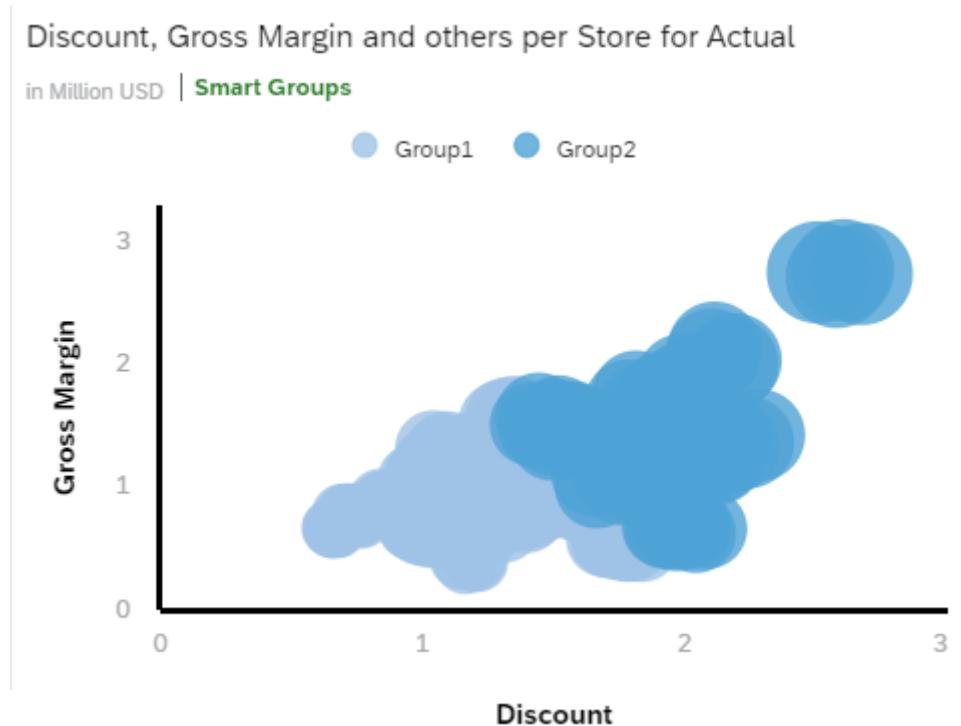


Figure 42 Bubble Chart with Smart Grouping Enabled

```

Chart_1.getDataSource().getResultSet();

// ResultSet[]
[{
  "@MeasureDimension": {
    "id": "[Account_BestRunJ_sold].[parentId].&[Discount]",
    "description": "Discount",
    "formattedValue": "1184733.37",
  }
}
  
```

```

    "rawValue": "1184733.37"
},
"Store_3z2g5g06m4": {
    "id": "ST1",
    "description": "Second Hand"
},
"Smart Group": {
    "id": "Predictive Clustering Group 1", // this data cell is in group 1
    "description": "Group1"
}
},
...
{
    "@MeasureDimension": {
        "id": "[Account_BestRunJ_sold].[parentId].&[Discount]",
        "description": "Discount",
        "formattedValue": "1969249.25",
        "rawValue": "1969249.25"
    },
    "Store_3z2g5g06m4": {
        "id": "ST38",
        "description": "Henrys Corner Store"
    },
    "Smart Group": {
        "id": "Predictive Clustering Group 2", // this data cell is in group 2
        "description": "Group 2"
    }
},
{
...
}]

```

5.8.1.4 Getting Dimension Attributes

With the Result Set script API, dimension attributes are sub-properties that complement the definition of dimensions and allow you to analyze data from a new perspective. If a dimension has attributes, then they're displayed as properties in each `ResultSet` element.

Example 14: Dimension 0BC_REG has some attributes

```

Table_1.getDataSource().getResultSet();
// ResultSet[]
[{
    "@MeasureDimension": {
        "id": "0002TLJDAAI840CJLDKA0280K",
        "description": "0BC_AMT",
        "parentId": undefined,
        "rawValue": "49996.6",
        "formattedValue": "DM49,996.60"
    },
    "0BC_REG": {
        "id": "AUSNRD",

```

```

    "description": "AUS/NRD",
    "parentId": undefined,
    "properties": {
        "OBC_REG.MIDDLE_TEXT": "AUS/NRD",
        "OBC_REG.DISPLAY_KEY_NC": "NRD"
    }
},
...
{
    "@MeasureDimension": {
        "id": "0002TLJDAAI840CJLDKA0280K",
        "description": "OBC_AMT",
        "parentId": undefined,
        "rawValue": "3649995.68"
        "formattedValue": "DM3,649,995.68"
    },
    "OBC_REG": { // no properties are returned here as the
                  // total value doesn't have properties
        "id": "SUMME",
        "description": "Totals",
        "parentId": undefined
    }
}
]

```

KIT_OBOC_TEST_DATATYPES_1

OBC_REG	OBC_REG (MIDDLE...)	OBC_REG (DISPLAY_KEY_NC)	Description	Measures		OBC_AMT	OBC_MAX
				ID			
AUS/NRD	AUS/NRD	NRD	AUS/NRD	AUS/NRD	DM49,996...	—	—
DE/BAW	DE/BAW	BAW	DE/BAW	DE/BAW	DM1,102,7...	1,134.620 KG	—
DE/BAY	DE/BAY	BAY	DE/BAY	DE/BAY	DM1,068,7...	1,183.430 KG	—
DE/SAC	DE/SAC	SAC	DE/SAC	DE/SAC	DM1,428,5...	1,169.430 KG	—
Totals					DM3,649,9...	1,183.430 KG	—

Figure 43 Get Dimension Attributes

5.8.1.5 Retrieving Visible Dimension Properties

You can retrieve additional dimension properties, the visible dimension properties, that were configured in the *Builder* panel at design time using the Data Source script API.

Note: Currently, this is supported only by data sources of Table widgets.

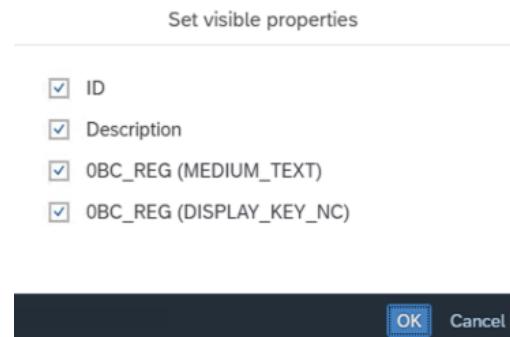


Figure 44 Visible Properties

The method `getResultSet()` returns an array of `ResultSet` objects. Each `ResultSet` is an object of `DataContext` objects. A `DataContext` object can contain the property `properties` that contains the visible properties. Or more formally:

```
Table_1.getDataSource().getResultSet(selection?: Selection | Selection[] | SelectionContext, offset?: integer, limit?: integer): ResultSet[]

// Property "properties" contains information on the visible properties
DataContext {
  id: string;
  description: string;
  parentId?: string;
  formattedValue?: string;
  rawValue?: string;
  properties?: {
    [key: string]: [string]
  }
}
```

5.8.2 Using the `getDataSelections` API

The `getDataSelections()` script API method returns the key-value pair of each cell. Using this script API is quite like the `getResultSet` script API method, but information such as `description` and `parentId` aren't returned.

Function Summary

```
// Returns the selection of data cells
// Offset / limit should be not less than zero.
// If offset / limit is invalid or not set, all data is returned.
// If the selection doesn't specify any MeasureDimension, all measures
// are returned.

Chart_1.getDataSource().getDataSelections(selection?: Selection | Selection[] | SelectionContext, offset?: integer, limit?: integer): Selection[]
Table_1.getDataSource().getDataSelections(selection?: Selection | Selection[] | SelectionContext, offset?: integer, limit?: integer): Selection[]
```

Let's use the previous chart from example 1 with *Gross Margin per Location*.

Example 15: Get data selections and no input parameter is specified

```
// If no input parameter, all data points of Chart_1 are in selection array
Chart_1.getDataSource().getDataSelections();

// Selection[]
[{
  "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]"
}, {
  "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC1]"
}, {
  ...
}]
```

5.8.2.1 Specifying Input Parameter and Filter Result

The input parameter of the `getDataSelections()` script API method is the same as the one of the `getResultSet()` script API method.

Let's use the previous example 9 *Gross Margin per Location and Product*.

Example 16: Get data selections and input parameter is specified

```
// Product isn't specified in input parameter.
// Some data points of Chart_1 are in selection array.
Chart_1.getDataSource().getDataSelections([
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
  "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]"
});

// Selection[]
[{
  "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]"
}, {
  "Location_4nm2e04531": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]",
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC1]"
}, {
  ...
}]
```

5.8.3 Using the getResultMember API

To get member-specific information, we introduce the script API method `getResultMember()`. Users can pass the ID of one dimension or a `DimensionInfo` and a data selection as a parameter. As a result, a `ResultMemberInfo` object is returned which contains `id`, `description`, and `parentId`.

For a dimension which has attributes, the attributes will be provided as additional properties of the `ResultMemberInfo` object.

Function Summary

```
// Get member metadata, works for data cell and header cell
Chart_1.getDataSource().getResultMember(dimension: String | DimensionInfo,
selection: Selection): ResultMemberInfo | undefined

Table_1.getDataSource().getResultMember(dimension: String | DimensionInfo,
selection: Selection): ResultMemberInfo | undefined

ResultMemberInfo {
  id: string
  description: string
  parentId: string
  properties?: { // "properties" doesn't contain "id" or "description"
    [key: string]: [string]
  }
}
```

Example 17: Identify a table cell and return a result member

BestRunJuice_SampleModel in Million USD					
		Account	Gross Margin	Gross Margin	Discount
Location	Product				Discount
▼ SA1 California	➤ PC4 Alcohol		25.39		30.72
	➤ PC1 Carbonated Drinks		44.26		15.65
	➤ PC2 Juices		102.78		130.30
	➤ PC3 Others		1.05		1.06
CT1 Los Angeles	➤ PC4 Alcohol		7.62		9.52
	➤ PC1 Carbonated Drinks		12.86		4.73
	➤ PC2 Juices		28.21		35.52
	➤ PC3 Others		0.29		0.28
CT2 San Francisco	➤ PC4 Alcohol		3.22		3.91
	➤ PC1 Carbonated Drinks		5.34		1.84
	➤ PC2 Juices		10.96		14.06
	➤ PC3 Others		0.10		0.10
CT3 San Diego	➤ PC4 Alcohol		2.32		2.67
	➤ PC1 Carbonated Drinks		2.92		1.03
	➤ PC2 Juices		12.17		15.74

Figure 45 Identify a Table Cell and Return a Result Member

```
Table_1.getDataSource().getResultMember("Location_4nm2e04531", {
  "Location_4nm2e04531":
  "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
  "Product_3e315003an":
  "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]",
  "@MeasureDimension": "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]"}
```

```

});
```

```

// ResultMember
{
  "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[CT1]",
  "description": "Los Angeles",
  "parentId": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]"
}

```

Example 18: Identify a header member as input parameter

The below example identifies a header member instead of a single data cell.

```

Table_1.getDataSource().getResultMember("Location_4nm2e04531", {
  "Location_4nm2e04531":
  "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]"
});
```

```

// ResultMember
{
  "id": "[Location_4nm2e04531].[State_47acc246_4m5x6u3k6s].&[SA1]",
  "description": "California"
}

```

Example 19: Return undefined if more than one result set member found

If the input parameter can't identify a single result member, for example, both [California](#) and [Los Angeles](#) have an [Alcohol](#) product, this script API method returns [undefined](#).

```

Table_1.getDataSource().getResultMember("Location_4nm2e04531", {
  "Product_3e315003an": "[Product_3e315003an].[Product_Catego_3o3x5e06y2].&[PC4]"
});
```

```

// ResultMember
undefined

```

If the same logic is applied to the table below, [California](#) is returned, as there is only one location in that table.

BestRunJuice_SampleModel in Million USD 1 Filter				
	Account	Gross Margin	Gross Margin	Discount
Location	Product			
➤ SA1 California	➤ PC4 Alcohol		25.39	30.72
	➤ PC1 Carbonated Drinks		44.26	15.65
	➤ PC2 Juices		102.78	130.30
	➤ PC3 Others		1.05	1.06

Figure 46 Return "undefined" If More Than One Result Set Member Found

Example 20: Dimension attribute is displayed as ResultMemberInfo properties

If a dimension has attributes, these attributes will be provided as properties.

```
Table_1.getDataSource().getResultMember("0BC_REG", {
    "0BC_REG": "AUSNRD",
    "@MeasureDimension": "0002TLJDAAI840CJLDKA0280K"
});

// below is the returned ResultMember
{
    "id": "AUSNRD",
    "description": "AUS/NRD",
    "parentId": undefined,
    "properties": {
        "0BC_REG.DISPLAY_KEY_NC": "NRD",
        "0BC_REG.MEDIUM_TEXT": "AUS/NRD"
    }
}
```

				Measures	OBC_AMT	OBC_MAX
OBC_REG	<i>OBC_REG (MIDDLE...)</i>	<i>OBC_REG (DISPLAY_KEY_NC)</i>	<i>Description</i>	<i>ID</i>		
AUS/NRD	<i>AUS/NRD</i>	<i>NRD</i>	<i>AUS/NRD</i>	<i>AUS/NRD</i>	DM49,996....	—
DE/BAW	<i>DE/BAW</i>	<i>BAW</i>	<i>DE/BAW</i>	<i>DE/BAW</i>	DM1,102,7...	1,134.620 KG
DE/BAY	<i>DE/BAY</i>	<i>BAY</i>	<i>DE/BAY</i>	<i>DE/BAY</i>	DM1,068,7...	1,183.430 KG
DE/SAC	<i>DE/SAC</i>	<i>SAC</i>	<i>DE/SAC</i>	<i>DE/SAC</i>	DM1,428,5...	1,169.430 KG
Totals					DM3,649,9...	1,183.430 KG

Figure 47 Attributes Displayed as ResultMemberInfo Properties

5.8.3.1 Retrieving Visible Dimension Properties

You can retrieve additional dimension properties, the visible dimension properties, that were configured in the *Builder* panel at design time using the Data Source script API.

Note: Currently, this is supported only by data sources of Table widgets.

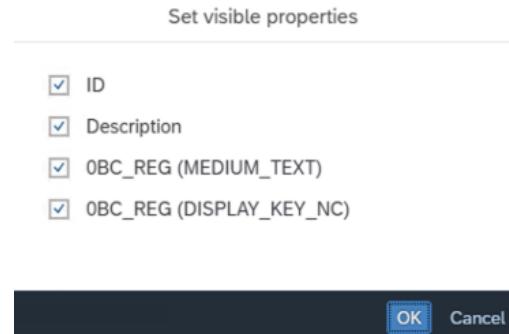


Figure 48 Visible Properties

The method `getResultMember()` returns an `ResultMemberInfo` object. It can contain the property `properties` that contains the visible properties. Or more formally:

```
Table_1.getDataSource().getResultMember(dimension: String | DimensionInfo,
selection: Selection) : ResultMemberInfo
```

```
// Property "properties" contains information on the visible properties
ResultMemberInfo {
    id: string;
    description: string;
    parentId: string;
    properties?: {
        [key: string]: [string]
    }
}
```

5.9 Widget Level Refresh Data API

By leveraging the Refresh Data script API, you can allow end users to trigger a data refresh for a specific widget or all widgets related to a data source of an application.

Refresh Data for All Widgets Related to a Data Source of an Application

To do this, you can write a script with the following script API:

```
Application.refreshData(datasources: [DataSource]);
```

After calling `Application.refreshData()`, the next line of script will wait until it finishes to run, to ensure the application gets the latest data.

Refresh Data for a Specific Widget

To do this, you can write a script with the following script API:

```
Widget.getDataSource().refreshData();
```

The widget should be a chart or table. Currently R visualization, Geo, and custom widgets aren't supported.

Note: Even if there are some widgets, for example charts created based on the same data source, refreshing one chart won't cause the other charts to refresh automatically.

Use Case 1: Refresh a Table or Chart When Initializing an Application

Write the `onInitialization` event script of the Canvas to refresh `Chart_1` and `Table_1` when initializing an application:

```
var ds1 = Chart_1.getDataSource();
var ds2 = Table_1.getDataSource();
Application.refreshData([ds1, ds2]);
```

Use Case 2: Refresh a Widget Periodically

Use the Refresh Data script API together with the Timer script API to refresh a widget periodically:

```
// Write the script for the onTimeout event of Timer_1
// to refresh data of Chart_1 and Chart_2 every minute.
Chart_1.getDataSource().refreshData();
Chart_2.getDataSource().refreshData();
Timer_1.start(60);
```

5.10 Prompt API

You can use the Prompt script API on a data source to perform variable-related operations in a script.

5.10.1 Opening the Prompt Dialog

You can open the Prompt dialog on a data source with the method `openPromptDialog()`.

Example:

In the following example, the Prompt dialog of a table's data source is opened:

```
Table_1.getDataSource().openPromptDialog();
```

5.10.2 Getting Variables

You can get the variables of a data source with the method `getVariables()`. This method returns an array of all variables as `VariableInfo` objects.

Example:

In the following example, the names of all variables of a data source are printed to the browser console:

```
var aVariables = Table_1.getDataSource().getVariables();
for (var i = 0; i < aVariables.length; i++) {
    console.log(aVariables[i].id);
}
```

5.10.3 Setting Variable Values

To set variable values, use the script API method `setVariableValue()` in the following form on a data source:

```
dataSource.setVariableValue(variable_name, variable_value);
```

Tip: In the script editor, there is context assist available for selecting variable names and variable values.

Note: By default, this function will apply variable values of a variable to the model used by the data source of the application. The widget can be configured such that variables are applied to the model of the widget only (see figure below).

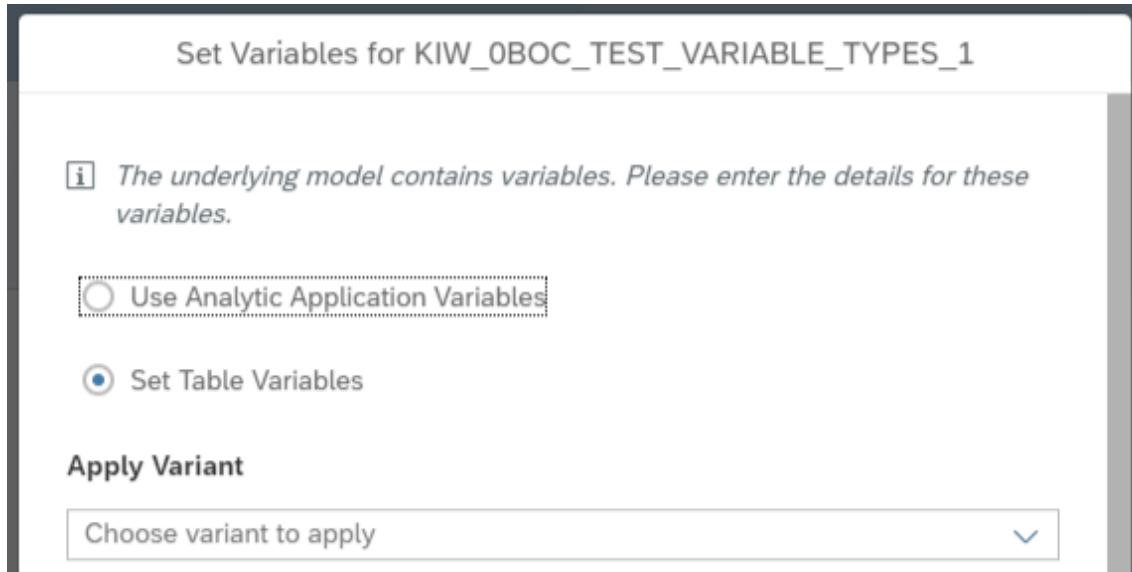


Figure 49: Prompt Dialog: Variable Values Are Applied to the Widget Only

You can find out, for example, in the title area of the table whether the variables are applied on the model of the data source of the application (grey braces) or on the model of the widget only (blue braces) (see figure below).

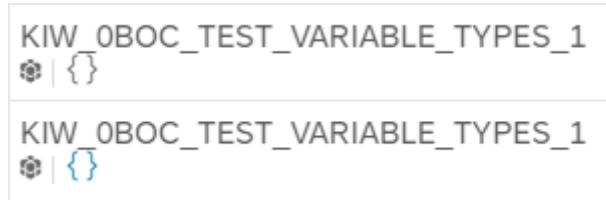


Figure 50: Variable Values Are Applied to the Model of the Application or the Widget

Note: This method isn't validating the specified variable values neither at runtime nor at design time. All values and value combinations which are accepted in the Prompt dialog will be supported. All other combinations might lead to errors or inconsistent state.

5.10.3.1 Single Variable Values

If the variable supports single variable values, you can set a variable value as follows:

Example:

```
Table_1.getDataSource().setVariableValue("VAR_NAME", {value: "5"});
```

or, alternatively,

```
Table_1.getDataSource().setVariableValue("VAR_NAME", "5");
```

If the variable supports excluding a single variable value, you can set the variable value as follows:

Example:

```
Table_1.getDataSource().setVariableValue("VAR_NAME", {exclude: true, value: "5"});
```

5.10.3.2 Multiple Variable Values

If the variable supports multiple values, you can set the variable values as follows:

Example:

```
Table_1.getDataSource().setVariableValue("VAR_NAME", {values: ["5", "7"]});
```

If the variable supports excluding multiple values, you can set the variable value as follows:

Example:

```
Table_1.getDataSource().setVariableValue("VAR_NAME", {exclude: true, values: ["5", "7"]});
```

5.10.3.3 Comparisons

If the variable supports comparison operations <, <=, >, and >= you can set the variable value as follows:

Example:

```
Table_1.getDataSource().setVariableValue("VAR_NAME", {less: "5"});
Table_1.getDataSource().setVariableValue("VAR_NAME", {lessOrEqual: "5"});
Table_1.getDataSource().setVariableValue("VAR_NAME", {greater: "5"});
Table_1.getDataSource().setVariableValue("VAR_NAME", {greaterOrEqual: "5"});
```

5.10.3.4 Ranges

If the variable supports a range of variable values, you can set the variable value as follows:

Example:

```
Table_1.getDataSource().setVariableValue("VAR_NAME", {from: "5", to: "7"});
```

If the variable supports excluding a range of variable values, you can set the variable value as follows:

Example:

```
Table_1.getDataSource().setVariableValue("VAR_NAME", {exclude: true, from: "5", to: "7"});
```

5.10.4 Getting Variable Values

You can get the values of a variable with the method `getVariableValues()`. It returns an array of all variable values as `VariableValue` objects. The script API definition is as follows:

```
getVariableValues(variable: string | VariableInfo): VariableValue[]
```

Example:

In the following example, the variable values of the variable `V_Country` are retrieved:

```
var values = Table_1.getDataSource().getVariableValues("V_Country");
```

Each value in the array is an instance of either `SingleVariableValue`, `MultipleVariableValue`, or `RangeVariableValue`, which all inherit from `VariableValue`. To work with such an instance, that is, to access its type-specific properties, cast the instance to the corresponding type first, using the `type` property. The following example shows how:

```
var value = Table_1.getDataSource().getVariableValues("V_Country")[0];
switch (value.type) {
    case VariableValueType.Single:
        var singleValue = cast(Type.SingleVariableValue, value);
        console.log(singleValue.value); // you can access the 'value' property now
        break;
    case VariableValueType.Multiple:
        var multiValue = cast(Type.MultipleVariableValue, value);
        console.log(multiValue.values); // you can access the 'values' property now
        break;
    case VariableValueType.Range:
        var rangeValue = cast(Type.RangeVariableValue, value);
        console.log(rangeValue.from); // you can access the 'from' property now
        console.log(rangeValue.to); // you can access the 'to' property now
        // further range properties: 'less', 'lessOrEqual', 'greater', 'greaterOrEqual'
        break;
    default:
        break;
}
```

Note: What you set with `setVariableValue()` isn't necessarily the same that's returned from `getVariableValues()`.

Example:

If you've several options to set multiple values, then the following lines are equivalent:

```
Table_1.getDataSource().setVariableValue("V_Country", {values: ["DE", "FR"], exclude: true});
Table_1.getDataSource().setVariableValue("V_Country", [{value: "DE", exclude: true}, {value: "FR", exclude: true}]);
```

`getVariableValues()` always returns as few `VariableValue` objects as possible, grouping all single values with the same sign, that is, including or excluding, into one `values` object. In the example, an array with only one object is returned:

```
[{values: ["DE", "FR"], exclude: true}]
```

In SAP BW models you can define variables that correspond to user-modifiable settings like filters or hierarchies. For example, setting a variable value sets the exact same value as the filter on the related dimension.

Note: Synchronization in the other direction is **not** performed by `getVariableValues()`. Thus, the `getVariableValues()` still returns the same variable value, even though the user might have modified the filter in the meantime. To retrieve the current value of such special variables, use `getDimensionFilters()` instead. For hierarchy variables use `getHierarchy()`.

5.10.5 Removing Variable Values

You can remove the variable value of a variable of a data source with the method `removeVariableValue()`.

Note: If you remove a variable value from a mandatory variable, then this operation is ignored.

Example:

In the following example, the variable value of variable `V_Supervisor` is removed:

```
Table_1.getDataSource().removeVariableValue("V_Supervisor");
```

5.10.6 Copying Variable Values

You can copy the variable value from one, several, or all variables of a data source to another variable with the method `copyVariableValueFrom()`.

Note: If you copy an empty variable value to a mandatory variable then copying this variable value is ignored.

Note: If you copy a variable value to a data source of a widget that overrides variables and the variable is of type text, then copying this variable value is ignored.

Example:

In the following example, the variable value of variable `V_Country` is copied from data source 1 to data source 2:

```
var DS_1 = Table_1.getDataSource();
Table_2.getDataSource().copyVariableValueFrom(DS_1, "V_Country");
```

Example:

In the following example, the variable values of variables `V_Country` and `V_Supervisor` are copied from data source 1 to data source 2:

```
var DS_1 = Table_1.getDataSource();
Table_2.getDataSource().copyVariableValueFrom(DS_1, ["V_Country", "V_Supervisor"]);
```

Example:

In the following example, the variable values of all variables are copied from data source 1 to data source 2:

```
var DS_1 = Table_1.getDataSource();
Table_2.getDataSource().copyVariableValueFrom(DS_1);
```

5.11 Popup and Dialog

A Popup or Dialog is usually a small window on top of the main page of the application. It communicates information to the user or prompts them for inputs.

For instance, a Popup can show a description of the application, and another Popup can ask the user to perform configurations. Because the popup acts as a container widget, you can put any other widget into the popup, such as a table, button, or checkbox.

You can choose to design a popup starting from scratch. Start with an empty Canvas and have the flexibility to add whatever widget you want. You can enable the header and footer setting to turn the popup directly into a popup dialog that has a consistent look and feel compared to other dialogs in SAP Analytics Cloud stories.

5.11.1 Main Popup and Dialog API

close

```
close(): void
```

getTitle

Hides the popup.

```
getTitle(): string
```

open

Returns the title of the popup.

```
open(): void
```

setTitle

Shows the popup.

```
setTitle(title: string): void
```

Sets the title of the popup.

5.11.2 Button-Related Popup and Dialog API

isButtonEnabled

```
isButtonEnabled(buttonId: string): boolean
```

Returns whether the specified button in the footer of the popup is enabled.

isButtonVisible

```
isButtonVisible(buttonId: string): boolean
```

Returns whether the specified button in the footer of the popup is visible.

setButtonEnabled

```
setButtonEnabled(buttonId: string, enabled: boolean): void
```

Enables or disables the specified button in the footer of the popup.

setButtonVisible

```
setButtonVisible(buttonId: string, visible: boolean): void
```

Shows or hides the specified button in the footer of the popup.

onButtonClick

```
onButtonClick(buttonId: string)
```

Called when the user clicks one of the buttons in the footer of the popup.

5.11.3 Popup and Dialog Events

onButtonClick

```
onButtonClick(buttonId: string)
```

Called when the user clicks one of the buttons in the footer of the popup.

5.11.4 Known Restrictions of Popup and Dialog

Need to add at least two widgets to a popup to run the popup as designed

We recommend adding at least two widgets to a popup as widgets are the visualization of the popup. If no widgets are added, you won't see the popup displayed when you trigger it while running the analytic application. If only one widget is added, the height and width you set for the popup won't take effect.

When a table or chart in the Canvas act as the source widget of a filter line widget in a popup, source widget can't find the filter line as its reference after reloading the analytic application

In the case when a table or chart in the Canvas act as the source widget of a filter line widget in a popup and you reopen or refresh the analytic application, you'll find the filter line isn't listed in the reference list of the table or chart widget after you choose *Find Reference*. This is because currently we don't initiate the filter line widget in the popup when you first entering an analytic application.

To solve this, for now we recommend activating the popups by clicking on each of them. Then the reference list will display all relevant results.

5.12 Text Widget

Use the Text widget to add user-defined text to your application. The style of the text can be configured as usual. You could refer to sample *Show R Visualization result in Text*. The most frequently used usages, getting and setting texts, and adding dynamic text are demonstrated and explained.

5.12.1 Changing Text

In *Show R Visualization result in Text*, the total value of gross margin is dynamically updated in `Text_GrossMargin` when switching among locations. Via script API method `applyText()` you can customize the display text of a Text at runtime:

```
if (totalSum) {  
    Text_GrossMargin.applyText(totalSum.toString());  
} else {  
    Text_GrossMargin.applyText("loading...");  
}
```

The Text shows “*loading...*” until `totalSum` is valid.

The text style can be configured by each segment. In-place edit the text by double-clicking the Text input field of `Text_Title` in Canvas and config the style of description.

5.12.2 Adding Dynamic Text

Add a script variable as the source of dynamic text to a Text widget to automatically update the text based on the values. For example, in *Show R Visualization result in Text*, `ScriptVariable_Currency` is defined and used in `Text_Title`.

The script variable can be exposed as URL parameter if you switch on the option. For example, if you input `p_ScriptVariable_Currency=CNY` in the URL link, you’ll get the following:

Total of Gross Margin in CNY: 235036949.4

5.13 RSS Feed

Use the RSS feed widget to present relevant articles from an RSS feed alongside data and visualizations. Leverage the open script API to dynamically update the list of RSS feeds according to your actions. For example, show blogs relevant to your area of interest. The sample *Present relevant RSS articles* can be referred to for the most frequently used script API methods.

Configure Feeds

The RSS feeds in the widget can be updated dynamically at runtime via a script API when you select in `Chart_RSSCategory` in *Present relevant RSS articles*.

Example:

If you select *Business* in the chart, *BBC Business* is added in the list of feeds and selected by default after running the scripts below:

```
RssReader_Content.removeAllFeeds();
RssReader_Content.addFeed("BBCBusiness", "https://feeds.bbci.co.uk/news/business/rss.xml");
RssReader_Content.setSelectedFeed("https://feeds.bbci.co.uk/news/business/rss.xml")
```

5.14 R Visualization

Use the R Visualization widget to leverage R scripts. It allows you to build your own visualizations, do calculation, and more. Refer to sample *Show R Visualization result in Text* for the most frequently used script API methods.

In the script of R Visualization, you can define parameters to get input values or return results calculated in the script.

Example:

Configure the title of visualization R Visualization in *Show R Visualization result in Text* per location by input parameter:

```
RVisualization.getInputParameters().setString("titleParam", "Gross Margin of Oregon");
```

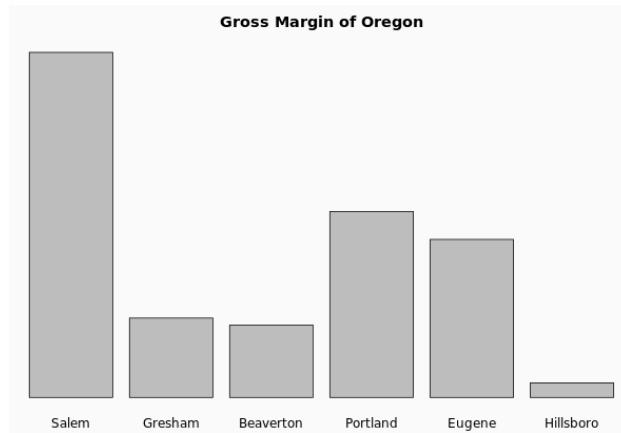
Example:

Calculate the total of gross margin in *RVisualization* script, and return the result:

```
RVisualization.getEnvironmentValues().getNumber("totalSum");
```

Configure the data source of the R Visualization via script API. For example, in *Show R Visualization result in Text*, the dimension filter is set to *Oregon* when you change location via *Dropdown_Location* by this

```
RVisualization.getDataFrame("BestRunJuice_SampleModel").getDataSource().setDimensionFilter("Location_4nm2e04531", ["CT13", "CT14", "CT15", "CT16", "CT17", "CT18"]);
```



5.15 Geo Map

The Geo Map widget is now supported in analytic applications. It lets application users overlay multiple layers of business data on a base map and explore the information behind the data from a geographical point of view.

The Geo Map widget in an analytic application has the same capabilities as in a story, and provides a script API to make changes by scripting.

Configure Layer Visibility

Since a Geo Map widget can have multiple visualization layers on the top, there is a script API to control their visibility so users can decide which layers they need to see.

```
GeoMap_1.getLayer(0).setVisible(true);  
GeoMap_1.getLayer(0).isVisible();
```

5.16 Timer

The Timer widget enables you to start a timer to trigger timing events. By leveraging the feature of a timer, you can realize different scenarios such as:

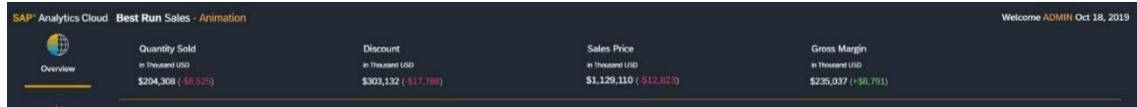
- Creating animations
- Sending notifications to end users regularly
- Refreshing your analytic application in a certain interval of time

To further demonstrate its usage, here are two samples for your reference.

5.16.1 Script API

```
Timer_1.start(delayInSeconds: number): void  
Timer_1.stop(): void  
Timer_1.isRunning(): boolean  
Timer_1.onTimeout // event
```

5.16.2 Sample 1 – Create Animation



In this sample, we add animation to the header above, making the tiles (widgets) shift from right to left repeatedly.

We use the Timer and Layout script APIs.

```
// Start a timer
Timer_1.start(ANIMATION_INTERVAL);

// To make the Widget moving, the Layout API is used to dynamically
// change the position of the widget.

// These are the 4 panels we want to apply animation to
PANELS = [Panel_10, Panel_11, Panel_12, Panel_13];
var numOfPanels = PANELS.length;
var moveStep = 0.1;

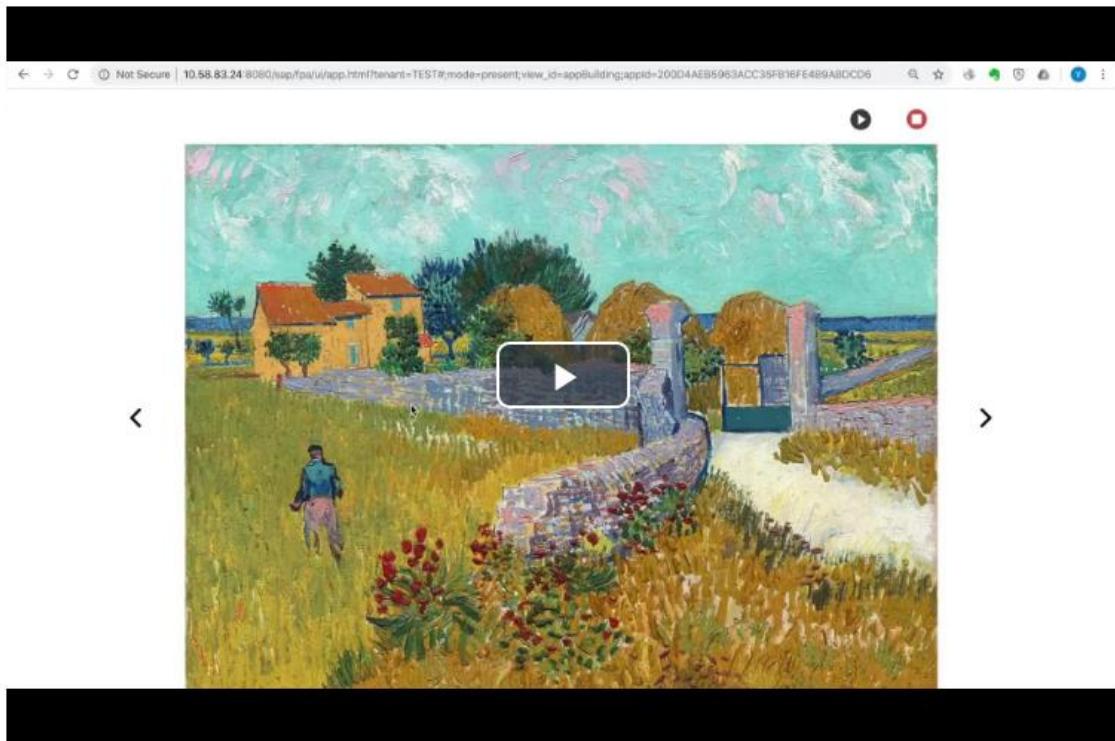
var firstPanel = PANELS[0];
var leftMarginOfFirstPanel = firstPanel.getLayout().getLeft().value;
var panelWidth = firstPanel.getLayout().getWidth().value;
var padding = 0;

if (leftMarginOfFirstPanel >= moveStep) {
    for (var i = 0; i < numOfPanels; i++) {
        var layout = PANELS[i].getLayout();
        layout.setLeft(LayoutValue.create(layout.getLeft().value - moveStep,
LayoutUnit.Percent));
    }
} else {
    // Move the first panel to end
    firstPanel.getLayout().setLeft(LayoutValue.create((panelWidth + padding)*
numOfPanels, LayoutUnit.Percent));
    for (i = 0; i < numOfPanels - 1; i++) {
        PANELS[i] = PANELS[i+1];
    }
    PANELS[i] = firstPanel;
    Util_Animation.doAnimation();
}
```

5.16.3 Sample 2 – Automatically Play the Application

This is an interesting requirement coming from customer. This customer wants an application that's displayed in a big screen with its pages automatically played in turn similar as a page book and can be manually stopped at will.

We can do it with Timer and TabStrip.



To make a TabStrip widget look like a page book, a small tip is to hide the header of the TabStrip, for example, using a shape, then use the script API method `TabStrip_1.setSelectedKey(TabID)` to dynamically “slide” the tab.

Then start a timer to repeat this action.

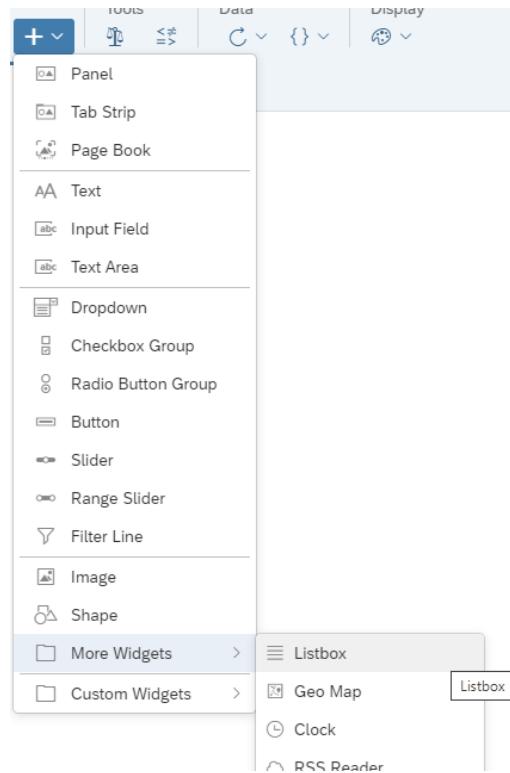
```
// Here's the code sample to switch and slide the tabs.  
var key = TabStrip_1.getSelectedKey();  
if (key === "Tab_1") {  
    TabStrip_1.setSelectedKey("Tab_2");  
} else if (key === "Tab_2") {  
    TabStrip_1.setSelectedKey("Tab_3");  
} else if (key === "Tab_3") {  
    TabStrip_1.setSelectedKey("Tab_1");  
}
```

5.17 List Box

The List Box widget displays several entries in a list. At runtime the end user can select one or more entries. This selection can be used, for example, as a filter.

5.17.1 Creating a List Box

Add a List Box from “+” in the tool bar.



5.17.2 Building a List Box

- Configure widget items via manual input or variable binding, which supports Script Variable, Tile Filter & Variable, and Model Variable.
- Set the selection mode single or multiple selection.
- (Optional) Configure the runtime write-back to a Script Variable.

List Box Value

Data Source Type: Script Variables

Binding ID with the variable: Please select

Binding Display Text with the variable: Please select

Listbox Properties

Enable the multi-selection: OFF

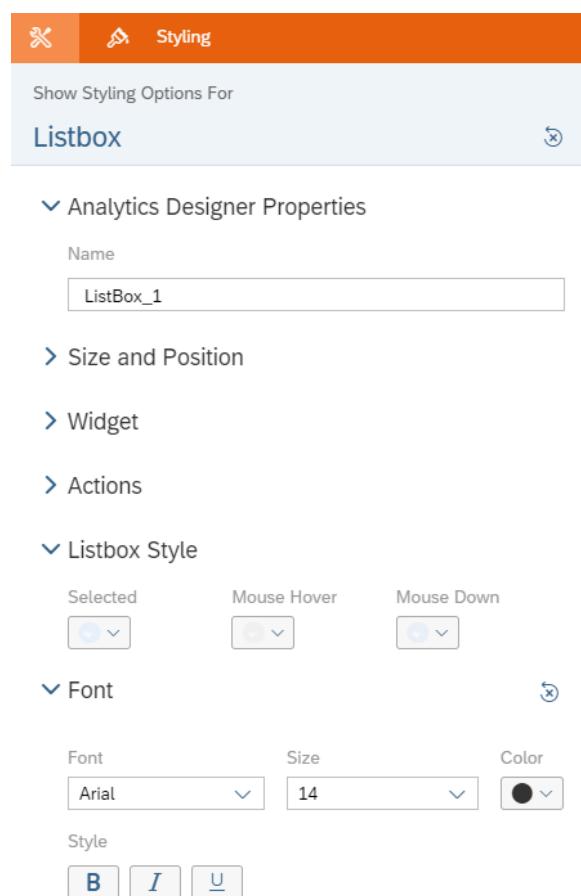
Enable the write-back in runtime: OFF

5.17.3 Configuring a List Box

You can configure the styling option of a List Box via its *Styling* panel. The available settings are:

- Analytics Designer Properties*

- *Name*
- *Size and Position*
- *Widget*
 - Background Color
 - Border
- *Actions*
 - Show at view time or not
 - Order
- *Listbox Style*
 - Selected
 - Mouse Hover
 - Mouse Down
- *Font*



5.17.4 Events

The `onSelect` event is triggered when the end user makes a selection at runtime.

5.17.5 Script API

Besides the following script API shared among all widgets:

```
// Returns the layout of the widget
```

```
getLayout(): Layout

// Returns whether the widget is visible
isVisible(): boolean

// Set to show or hide the widget
setVisible(visible: boolean): void

the List Box has the following specific script API:

// Adds a new item to the List Box.
// The item is specified by a key and an optional text.
addItem(key: string, text?: string): void

// Selects an item in the List Box. The item is specified by its key.
setSelectedKey(key: string): void

// Selects items in the List Box. The items are specified by their keys.
// If the List Box is single selection, the first item of the specified keys
// will be selected.
setSelectedKeys(key: string[]): void

// Returns the text of the selected item in the List Box.
getSelectedText(): string

// Returns the texts of the selected items in the List Box.
getSelectedTexts(): string[]

// Returns the key of the selected item in the List Box.
// If the List Box is multiple selection, the key of the first selected item
// will be returned.
getSelectedKey(): string

// Returns the keys of the selected items in the List Box.
getSelectedKeys(): string[]

// Removes an item from the List Box. The item is specified by its key.
removeItem(key: string): void

// Removes all items from the List Box.
removeAllItems(): void
```

5.17.6 Sample Use Case

```
// Example to add items and make selection in a List Box with multiple selection.
listBox_1.addItem("1", "Juices");
listBox_1.addItem("2", "Carbonated Drinks");
listBox_1.addItem("3", "Alcohol");
listBox_1.setSelectedKeys(["1", "2"]);
```

5.18 Layout API

You, as an analytic application developer, can directly set a widget's size and position in a parent container in the *Styling* panel. In addition to that, by leveraging the layout-related script API, you can allow application users to dynamically set a widget's size and position according to the application logic and window size.

```
LayoutUnit.Pixel // sets the unit of the layout as Pixel
LayoutUnit.Auto // sets the unit of the layout as Auto
LayoutUnit.Percent // sets the unit of the layout as Percent
LayoutValue.create(value: number, LayoutUnit: Unit) // sets the layout value by
creating a value with a certain unit

getLayout(): Layout // gets the layout of a widget
Layout.getLeft(): Unit; // returns the left margin between the widget that you
define layout for and the widget's parent container.
Layout.setLeft(value: Unit); // sets the left margin between the widget that you
define layout for and the widget's parent container.
Layout.getRight(): Unit; // returns the right margin between the widget that you
define layout for and the widget's parent container.
Layout.setRight(value: Unit); // sets the right margin between the widget that you
define layout for and the widget's parent container.
Layout.getTop(): Unit; // returns the top margin between the widget that you define
layout for and the widget's parent container.
Layout.setTop(value: Unit); // sets the top margin between the widget that you
define layout for and the widget's parent container.
Layout.getBottom(): Unit; // returns the bottom margin between the widget that you
define layout for and the widget's parent container.
Layout.setBottom(value: Unit); // sets the bottom margin between the widget that
you define layout for and the widget's parent container.
Layout.getWidth(): Unit; // returns the width of the widget that you define layout
for.
Layout.setWidth(value: Unit); // sets the width of the widget that you define
layout for.
Layout.getHeight(): Unit; // returns the height of the widget that you define
layout for.
Layout.setHeight(value: Unit); // sets the height of the widget that you define
layout for.

// Application Canvas Resize Event, the event is cached to be
// dispatched every 500ms when the application window resizes.
Application.onResize() = function() {
};

Application.innerHeight() // If Canvas' size is fixed, it returns the height of
the Canvas; if dynamic, returns the height of the viewport, the visible area of the
window.
Application.innerWidth() // If Canvas' size is fixed, it returns the width of
the Canvas; if dynamic, returns the width of the viewport, the visible area of the
window.
```

We don't have the mechanism yet to automatically flow the widgets when the screen size changes, which will be introduced in future. But we can cover some of the responsive scenarios by combining dynamic layout and script APIs. In an analytic application, more than just flow UI, you've the flexibility to add a widget on top of a background shape, overlapping but not flow them, and they can shrink or grow in the same proportion when the window size changes.

You need two steps to make it happen:

Step 1: Set Size and Position in Styling Panel

You can set each widget's *Left*, *Width*, *Right* and *Top*, *Height*, *Bottom* values in Pixel, Percentage and Auto (relative to its parent container, root Canvas if not in a container) values on the *Styling* panel's *Layout Section*.

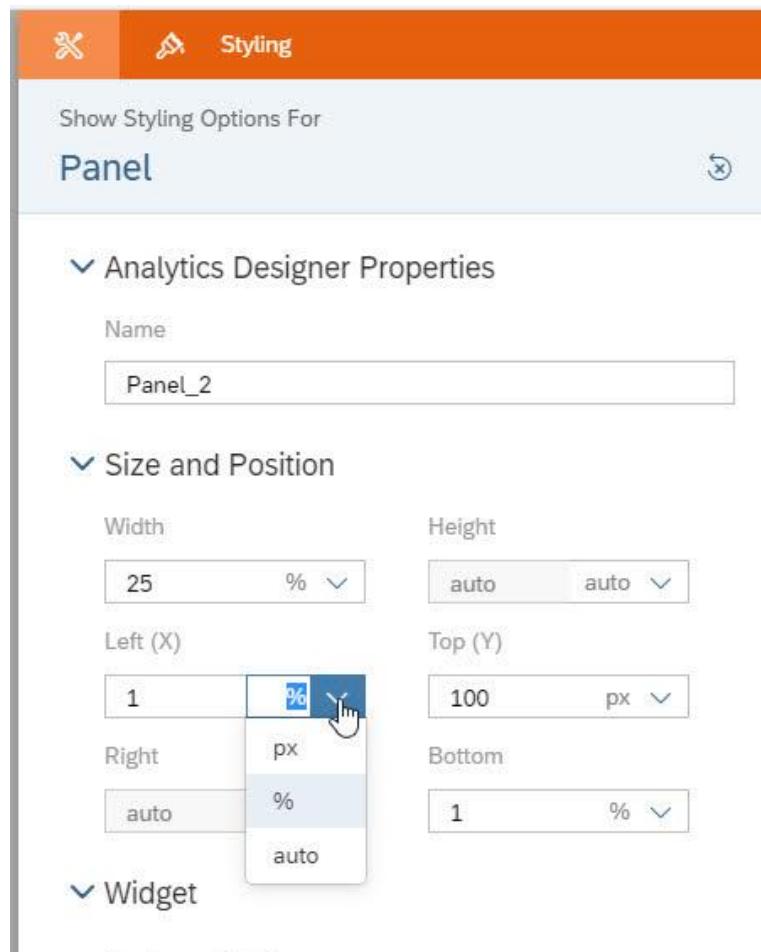


Figure 51: Layout Section in the *Styling* Panel

To adapt to the screen real-estate at runtime on different machines or browser window, you need to set the unit to percentage (%) or auto.

Step 2: Dynamically Set the Size and Position in Application.onResize Event

The application's `onResize` event is dispatched every 500 ms when the application window resizes.

Inside the `onResize` event script, you can use the Layout script API to dynamically set the size and position.

Below code sample shows how to adjust the layout to fit a small screen size like phone.

```
// small screen size
if (screenWidth < 500 || screenHeight < 500) {
    Panel_3.setVisible(false);
    Panel_2.setLayout().setWidth(LayoutValue.create(98,
        LayoutUnit.Percent));
    Panel_2.setLayout().setBottom(LayoutValue.Auto);
    Panel_2.setLayout().setHeight(LayoutValue.create(376,
        LayoutUnit.Pixel));

    Panel_3.setLayout().setBottom(LayoutValue.Auto);
    Panel_3.setLayout().setLeft(LayoutValue.create(1,
        LayoutUnit.Percent));
    Panel_3.setLayout().setTop(LayoutValue.create(476,
        LayoutUnit.Pixel));
    if (screenWidth < 500) {
        // one column
        Panel_3.setLayout().setHeight(LayoutValue.create(
            (baseChartHeight + padding) * 4 + padding * 3 +
            Table_1.getHeight().value, LayoutUnit.Pixel));
    } else {
        // two columns
        Panel_3.setLayout().setHeight(LayoutValue.create(840,
            LayoutUnit.Pixel));
    }
    Panel_3.setVisible(true);
}
```

With the Layout script API, you've all the flexibility to adjust the application based on the screen size, to create a responsive application in an analytic application.

5.19 Set Theme API

You, as an analytic application developer, can leverage the Set Theme script API to allow end users to flexibly change the theme of their applications when running an application.

To specify a theme, use the syntax `Application.setTheme()`, type `CTRL+Spacebar` and the theme selector will automatically pop up for you to choose the theme you want to apply from the *Files* repository. After you choose a theme, the corresponding theme ID will be displayed in the syntax. If you choose not to define a theme in the syntax, then the application will apply the default light theme instead.

Note: Currently, there's a restriction that calling the `setTheme()` script API method in a popup doesn't affect the theme settings in the popup. To solve this, you can add a panel to a popup and include all widgets in it. Then, when you trigger the `setTheme()` script API method in a popup, all widgets added to the panel will apply the new settings.

Here's an example that shows how to leverage the Set Theme script API to allow end users to switch between different themes for your application:

First add a dropdown widget `Theme_Dropdown` to the Canvas. In the *Builder* panel of the dropdown, fill the value column with the theme IDs and the text column with corresponding theme names.

Then write the following script for the drop-down:

```
var themeId = Theme_Dropdown.getSelectedKey();
Application.setTheme(themeId);
```

Now, when users run the application, they can select a theme from the dropdown list to change to a different theme.

In addition to use the Set Theme script API, you can also enable end users to apply a specific theme when loading an analytic application by directly adding the theme ID to the application's URL address. For example, like this:

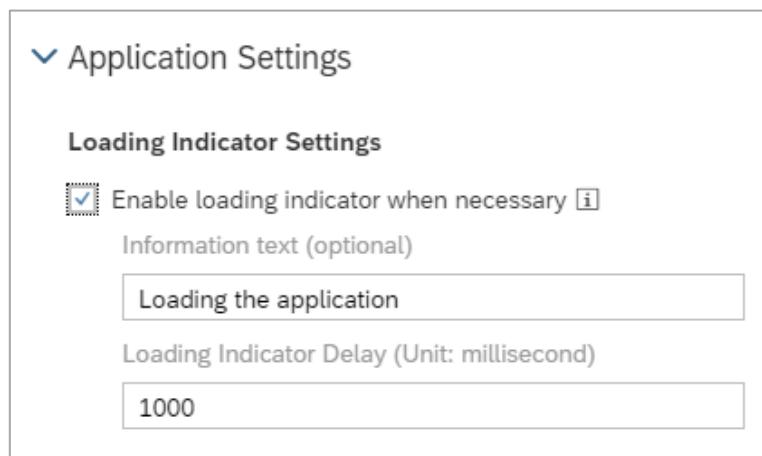
<https://master-app-building/sap/fpa/ui/bo/application/4FA12EC04829FDC682399273A7A3A0C?mode=embed;themeId=D991AAE EC518947626D749EDFF57D64C>

5.20 Loading Indicator

In some analytic applications, where widgets (for example, charts) constantly refresh, users can still interact with the data points during refreshing. This may make an error message show up. To address this issue, the analytic application can automatically display a loading indicator for long running scripts, as well as enable developers to show or hide the loading indicator via a script API on the application, a popup, or a container widget to block user interaction while other processing is still ongoing.

5.20.1 Automatic Loading Indicator

You can display an automatic loading indicator on the analytic application for long running scripts. The indicator will disappear automatically when the scripts are done.



In the *Styling* panel, you can enable or disable the automatic loading indicator.

Besides this switch, you can customize the following:

- an information text shown along with the indicator to let application users know what happens in the background.
- a time of delay (in milliseconds)

Besides the configuration in the *Styling* panel at design time, the automatic loading indicator can be also turned on and off via scripting:

```
// Enable/disable automatic loading indicator  
Application.setAutomaticBusyIndicatorEnabled(true);
```

5.20.2 Script API

Besides displaying an automatic loading indicator, you can show or hide the loading indicator via a script API as well.

The script API is available for the following objects: [Application](#), [Popup](#), and container widgets, such as [TabStrip](#) and [Panel](#). The text shown along with the indicator can be configured via an optional parameter.

```
// Show loading indicator, add text to loading indicator if text is specified  
Application.showBusyIndicator("Loading the application"); // cover the whole  
application page  
Popup_1.showBusyIndicator("Loading the popup"); // cover the popup  
TabStrip_1.showBusyIndicator("Loading the tab strip"); // cover the tab strip  
Panel_1.showBusyIndicator("Loading the panel"); // cover the panel  
  
// Hide loading indicator  
Application.hideBusyIndicator();  
Popup_1.hideBusyIndicator();  
TabStrip_1.hideBusyIndicator();  
Panel_1.hideBusyIndicator();
```

5.21 Bookmark API

You, as an analytic application developer, can use the Bookmark API to control bookmarks in analytic designer scripts.

5.21.1 Saving Bookmarks

You can save a bookmark with the [save\(\)](#) script API method.

Example:

In the following example, the state of the analytic application is captured in a bookmark with the bookmark name “application-bookmark”. The bookmark is available globally. The save operation overwrites any previous bookmark of this name:

```
BookmarkSet_1.save("application bookmark", true, true);
```

5.21.2 Saving Additional Properties with Bookmarks

When saving a bookmark, you can save additional information (properties) along with the bookmark.

You can accomplish this with the following APIs:

The [saveBookmark\(\)](#) script API method of the [BookmarkSet](#) component. It has the following signature:

```
BookmarkSet.saveBookmark(BookmarkSaveInfo: BookmarkSaveInfo, overwrite?: boolean):  
BookmarkInfo
```

Note: This method replaces the method `save()`, which is deprecated.

The `BookmarkSaveInfo` class, which is passed to `saveBookmark()` and contains properties as key-value pairs:

```
class BookmarkSaveInfo {  
    name: string,  
    isGlobal?: boolean,  
    isDefault?: boolean,  
    properties?: { [key: string]: string | undefined }  
}
```

The `Bookmark` class, which is returned by `saveBookmark()` and contains properties:

```
class BookmarkInfo {  
    id: string,  
    name: string,  
    displayName: string,  
    version: integer,  
    isGlobal: boolean,  
    isDefault?: boolean,  
    properties?: { [key: string]: string | undefined }  
}
```

The option `isDefault` indicates whether the bookmark is the default bookmark. The default bookmark is loaded when the analytic application's URL contains the URL parameter `bookmarkId=DEFAULT`.

Example:

In the following example, the model ID is saved along with the bookmark. This additional information is used to list all bookmarks which use the model ID "`modelId`":

```
// Create bookmark properties (key-value pairs)  
var oBookmarkProperty1 = { "modelId": "BestRunJuice" };  
var oBookmarkProperty2 = { "modelId": "BusinessPlanning" };  
  
// Save bookmark bk1 and bk2 with model ID in properties  
BookmarkSet_1.saveBookmark({ name: "bk1", "isGlobal": true, "properties":  
oBookmarkProperty1 }, true);  
BookmarkSet_1.saveBookmark({ name: "bk2", "isGlobal": true, "properties":  
oBookmarkProperty2 }, true);  
  
// List all bookmarks using the "BestRunJuice" model  
var aBookmarkInfo = BookmarkSet_1.getAll();  
for (var i = 0; i < aBookmarkInfo.length; i++) {  
    if (aBookmarkInfo[i].properties) {  
        for (var j = 0; j < aBookmarkInfo[i].properties.length; j++) {  
            if (aBookmarkInfo[i].properties[j] &&  
                aBookmarkInfo[i].properties[j]["modelId"] === "BestRunJuice") {  
                // print the name of bookmark which uses the "BestRunJuice" model  
                console.log(aBookmarkInfo[i].name);  
            }  
        }  
    }  
}
```

```
    }  
}  
}
```

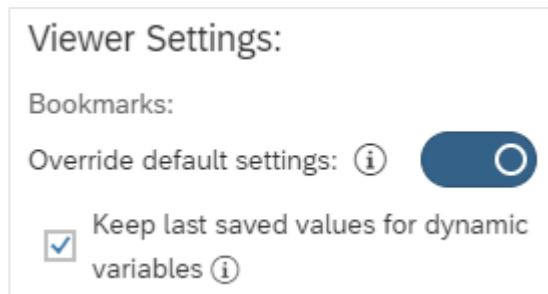
5.21.3 Keeping Last Saved Values of Dynamic Variables in Bookmark

You can now use the default values of dynamic variables retrieved from BW backend when opening the application for the first time and save the last changed values in bookmark afterwards.

To accomplish this, you can define whether the last changes to the dynamic variable values can be saved with bookmarks via *Analytic Application Details* settings or script API.

Settings in Analytic Application Details

Switch on *Override default settings* and select *Keep last saved values for dynamic variables* to modify the default variable behavior carried from the model level. Application users can therefore save a bookmark that will be opened with the last changed values of dynamic variables.



Script API

A new flag of keeping last changed variable value is added to the existing `saveBookmark` API.

```
// saveBookmark API  
saveBookmark(bookmarkInfo: BookmarkSaveInfo, override?: boolean):  
Promise<BookmarkInfo | undefined>  
  
// new flag isKeepLastDynamicVariableValue in BookmarkSaveInfo  
export interface BookmarkSaveInfo {  
    id: string;  
    name: string;  
    displayName: string;  
    version: number;  
    isGlobal: boolean;  
    isDefault?: boolean;  
    properties?: { [key: string]: string | undefined };  
    isNativeOptimized?: boolean;  
    // Keep last changed value of dynamic variable or not  
    isKeepLastDynamicVariableValue?: boolean;  
}
```

5.21.4 Loading Bookmarks

You can load a bookmark at runtime, also called “to apply a bookmark”, instantly via scripting and keep the existing widget states from the users’ previous interactions for any not-included widgets (for example, selections, filters, drilldowns, sortings, rankings, expanding and collapsing states of charts and tables, input values of input fields or sliders).

When applying a bookmark, the operation is applied to the bookmark’s *Included Components* configured at design time.

```
// Apply the bookmark to the current analytic application. Returns false if the
// bookmark isn't available to the current user. The input is the bookmark ID or a
// BookmarkInfo object.
BookmarkSet_1.apply(bookmark);
```

When applying a bookmark in an analytics designer script at runtime, then the application is reloaded.

5.21.4.1 Tip: Loading Bookmarks in the onInitialization Event Script

When you load a bookmark at runtime with the `apply()` script API method in the `onInitialization` event script, then the analytic application is reloaded and the `onInitialization` event script is executed again. This can lead to an infinite loop. The following example shows one way to avoid this:

Example:

In the following snippet from an `onInitialization` event script, the first bookmark is applied to the analytic application without causing an infinite loop:

```
var bookmarks = BookmarkSet_1.getAll();
if ((BookmarkSet_1.getAppliedBookmark() === undefined) && (bookmarks.length > 0)) {
    var bookmark = bookmarks[0];
    BookmarkSet_1.apply(bookmark);
}
```

5.21.5 Getting Bookmark Information

You can retrieve information about bookmarks with the `getAll()`, `getVersion()`, and `getAppliedBookmark()` script API methods. You can check if the state of the current application is different from its state captured in a bookmark with the `isSameAsApplicationState()` script API method.

Example:

In the following example, see some applications of the described script API methods:

```
BookmarkSet_1.getAll();      // get all valid bookmarks
BookmarkSet_1.getVersion(); // get the version of current bookmark

// get current applied bookmark
var bookmarkInfo = BookmarkSet_1.getAppliedBookmark();

// check if bookmark is changed
BookmarkSet_1.isSameAsApplicationState(bookmarkInfo);
```

5.21.6 Deleting Bookmarks

You can delete a bookmark with the `deleteBookmark()` script API.

Example:

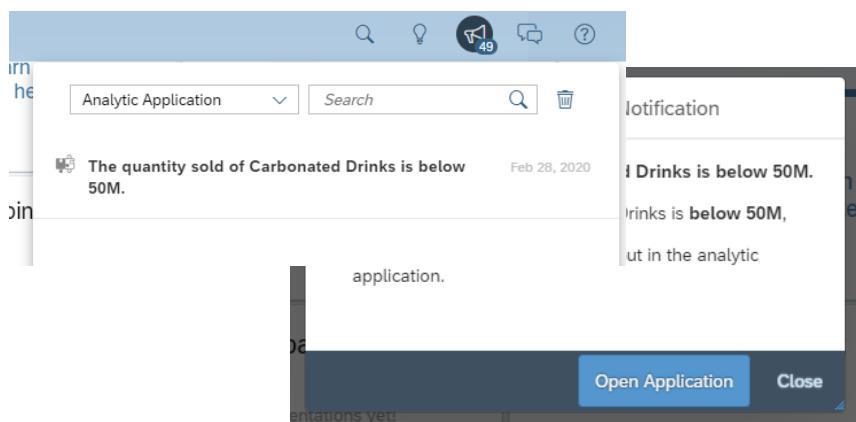
In the following example, the currently applied bookmark is deleted:

```
var bookmarkInfo = BookmarkSet_1.getAppliedBookmark();
BookmarkSet_1.deleteBookmark(bookmarkInfo); // delete bookmark
```

5.22 Notification API

When an analytic application is running at the backend, for example, when scheduled for publishing, you can use a script API to send notifications to the application consumers in the *Notification* panel and via e-mails.

Notification Panel



Email template



The quantity sold of Carbonated Drinks is below 50M.

Dear User

The quantity sold of Carbonated Drinks is below 50M, which is below estimation. For more details, please check it out in the analytic application.

[Open Analytic Application](#)

Notifications include the following elements:

- Title

- Body content (in HTML format with limited support, including the tags `<i>`, ``, `<u>`, `<a>`, and `
`)
- Action button to open the analytic application

You can also configure in addition the following elements:

- Recipients (value help available)
- View mode of the analytic application at runtime
- URL parameters as key-value pairs
- Whether to send an e-mail in addition to the notification or not
- Whether to send the notification to SAP Analytics Cloud's iOS Mobile App

The script API is defined as follows:

```
enum ApplicationMode {
    Present,
    Embed,
    View
}

NotificationOptions {
    title: string,
    content?: string,
    receivers?: string[],           // default: current application user
    mode?: ApplicationMode,        // default: Present
    parameters?: UrlParameter[],   // array, no optional single URL parameter
    isSendEmail?: boolean,          // default: false
    IsSendMobileNotification?: boolean // default: false
}

Application.sendNotification(notification: NotificationOptions): boolean
```

The following example shows a sample use case:

```
/*
Example: Will send both notification and e-mail to user (Jack).
Notification title is 'notification title'. Notification content is "notification
content".
*/
var param1 = UrlParameter.create("bookmarkId", "17889112-4619-4801-8954-
637818290511");
Application.sendNotification({
    "title": "notification title",
    "content": "notification content",
    "receivers": ["Jack"],
    "mode": ApplicationMode.Present,
    "parameters": [param1],
    "isSendEmail": true});
```

5.23 Move Widget API

You can use the following script API to move widgets to the Canvas, a panel, or a tab strip:

```
Application.moveWidget(widget: Widget);
TabStrip.moveWidget(tabName: string, widget: Widget);
Panel.moveWidget(widget: Widget);
```

Examples:

```
// move widget to container

// move button to Canvas
Application.moveWidget(Button_1);

// move button to TabStrip_1.Tab_1
TabStrip_1.moveWidget("Tab_1", Button_1);

// move button to Panel_1
Panel_1.moveWidget(Button_1);

// move container to another container

// move TabStrip_2 to TabStrip_1.Tab_1
TabStrip_1.moveWidget("Tab_1", TabStrip_2);

// move Panel_2 to Panel_1
Panel_1.moveWidget(Panel_2);

// the below cases won't work
TabStrip_1.moveWidget("Tab_1", TabStrip_1);
Panel_1.moveWidget(Panel_1);
```

Popup

There is no script API to move a widget to a popup, but you can add a container (panel or tab strip) to the popup and move the widget to this container.

You can't move a widget, whose parent is a popup, to other containers. For example, consider this containment hierarchy: `Popup_1` – `Panel_1` – `Button_1`. You can't move `Panel_1` to another container on the Canvas, but you can move `Button_1`.

Bookmark

You can bookmark the state after the `moveWidget()` script API call.

This is what will happen in one special case:

Example:

`Button_1` is in the Canvas.

Run the application. Move `Button_1` to `Panel_1` via the `moveWidget()` script API method. Save the bookmark.

Open the application again, move `Button_1` to `Panel_2` and delete `Panel_1`. Save the application.

Open the saved bookmark. `Button_1` is in `Panel_2`.

In this case, we recommend that the application developer should update the bookmark version.

5.24 Property Binding of Simple Widgets

Besides setting values of simple widgets manually, you can bind their values to primitive-type Script Variables, Tile Filters or Variables, Model Variables, and so on. This updates the value of the widgets dynamically. The supported simple widgets include the following:

- Checkbox Group
- Dropdown
- Image
- Input Field
- List Box
- Radio Button Group
- Range Slider
- Slider
- Text Area

A value that an end user selected or updated at runtime can be written back to a specific Script Variable as well.

Note: This isn't supported for the Image widget.

5.24.1 Binding the Value of a Simple Widget

A simple widget's value can be bound to one of the following:

- a primitive-type Script Variable
- an Application Property
- a Tile Filter & Variable
- a Model Variable.

The supported types vary with each widget.

Dropdown, CheckboxGroup, RadioButtonGroup

- Bindable values: ID, Display Text
- Supported bindings: Script Variable, Tile Filter & Variable, Model Variable

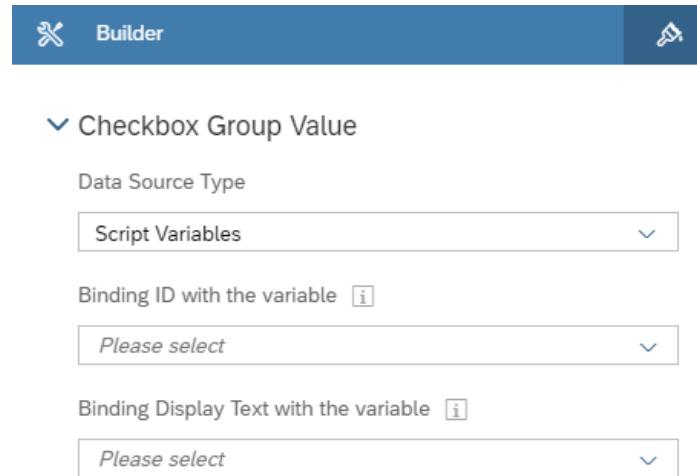


Figure 52: Bindings of Checkbox Group

InputField, TextArea

- Bindable values: Text
- Supported bindings: Script Variable, Tile Filter & Variable, Model Variable, Application Property (incl. Current User/Time/Date, Last Modified Date, Last Modified Date/Time, Last Modifier, Creator)

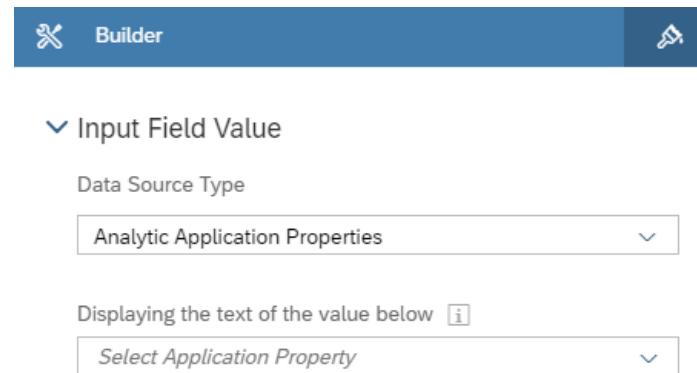


Figure 53: Bindings of Input Field

Slider, RangeSlider

- Bindable values: (Slider) Current value; (Range Slider) Start value, End value
- Supported Bindings: Script Variable, Tile Filter & Variable, Model Variable

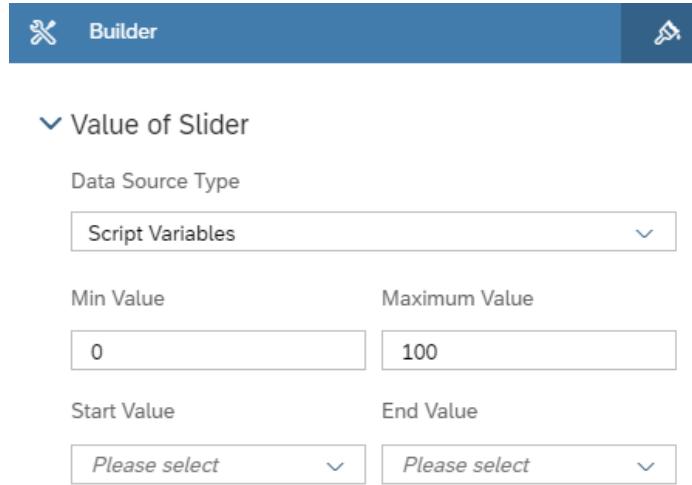


Figure 54: Bindings of Slider

Image

- Bindable values: Image URL
- Supported Bindings: Script Variable

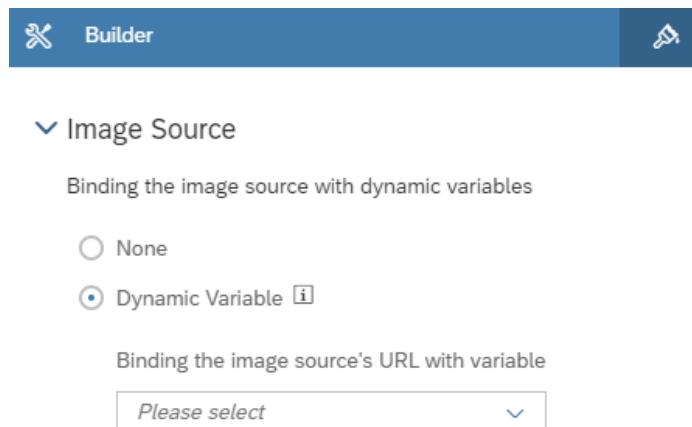


Figure 55: Bindings of Image

5.24.2 Enabling Write-Back at Runtime

A value that an end user selected or updated at runtime can be passed to a Script Variable, which may be referred to in scripts or by other widgets.

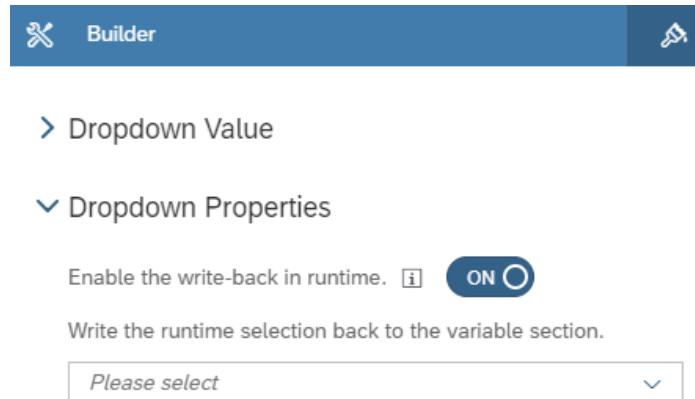


Figure 56: Write-Back Bindings of Dropdown

5.25 Get Application Info API

You can use this script API to get the information of one specific analytic app, including its ID, name, and description.

Example:

In the following example, a customer has three SAP Analytic Cloud (SAC) tenants and three 3 SAP BW landscapes (Dev, Acceptance, and Production). The customer wants the SAC analytic applications to check from which SAC tenant an analytic application has been launched wants to open the WebIntelligence report on the corresponding SAP BW landscape with the help of the Hyperlink functionality of the Navigation script API:

Example:

```
var info = Application.getInfo();
// output
{
  "id": "F16A62E886D31E5E56150FFD18396CA1",
  "name": "applicationInfoExample",
  "description": "application info desc"
}
var appId = info.id;
var url = NavigationUtils.createApplicationUrl(appId);
// output
"Error! Hyperlink reference not valid."
```

5.26 Application Messages API

You can use a script API to create a toast message to inform the end user.

You can also use the script API to suppress system messages (success, warning, info) in some cases in order not to annoy end users. Fatal errors and reload can't be suppressed because it's critical to let end users know. Messages will be written to the console even if they're suppressed by `setMessageTypesToShow()` for supportability reasons.

Script API

```
enum ApplicationMessageType {  
    Success,  
    Info,  
    Warning,  
    Error  
}  
  
// create toast message to inform end users  
Application.showMessage(messageType: ApplicationMessageType, message: string): void  
  
// show certain types of messages, others will be suppressed  
Application.setMessageTypesToShow(messageTypes: [ApplicationMessageType]): void
```

5.27 Gestures on Mobile Devices

You can create analytic applications for end users on mobile devices to use the shake gesture. The `onOrientationChange` event is now supported as well. On the Button, Shape, and Image widgets end users can also use the long-press gesture.

New events

Application:

```
// triggered before Application.onResize() event  
onOrientationChange(angle: DeviceOrientation)  
onShake()
```

Button, Image, Shape:

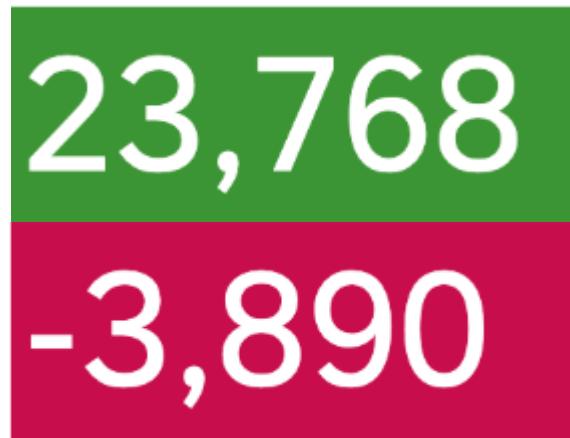
```
onLongPress()
```

5.28 Set Widget Style API

You can set widget styles at runtime.

For example, you can set the background color based on a value.

Without this script API, a trick to achieve this was to create two text widgets, each with a red or green background color. Then `setVisible()` was used to control which text widget should be shown. This increased the number of widgets.



Script API

```
Text.setStyle({
    backgroundColor: value,
    color: value
}): void

InputField.setStyle({
    backgroundColor: value,
    borderColor: value,
    color: value
}): void

TextArea.setStyle({
    backgroundColor: value,
    borderColor: value
}): void

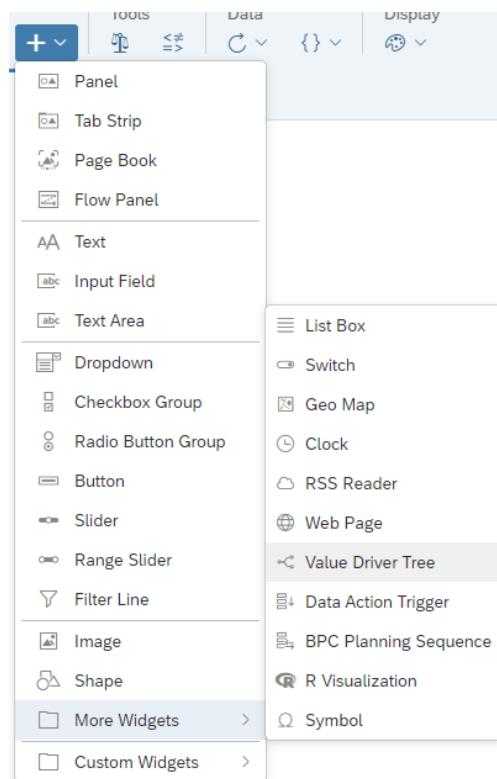
Shape.setStyle({
    fillColor: value,
    lineColor: value
}): void
```

5.29 Value Driver Tree

With the Value Driver Tree widget, you can do budgeting, forecasting based on different scenarios, and make daily business decisions.

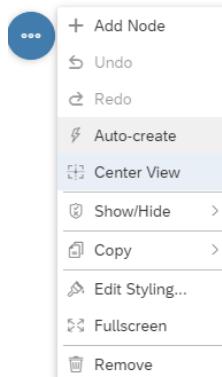
Create a Value Driver Tree

Add a Value Driver Tree widget from the “+” icon in the toolbar.

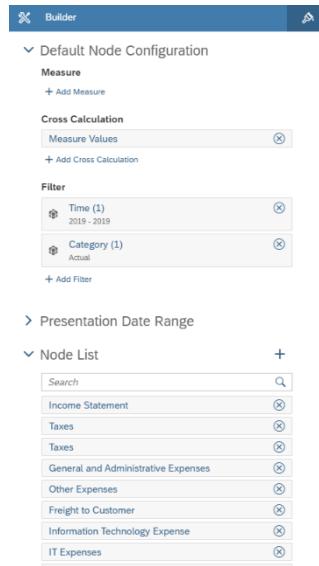


Build a Value Driver Tree

- Auto-create the tree based on the data by selecting *Auto-create* from the context menu.



- Configure nodes, presentation date range, and so on in the *Builder* panel.

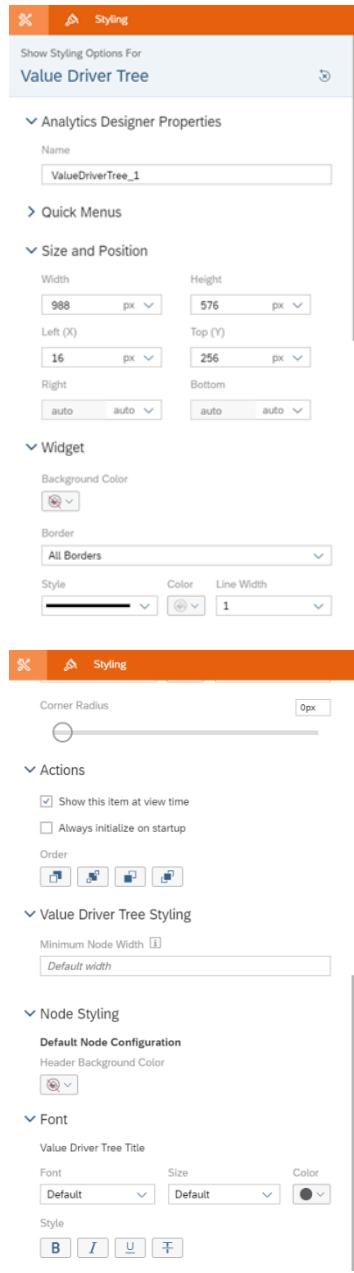


Configure a Value Driver Tree

Users can configure the styling options of a Value Driver Tree widget via its *Styling* panel.

The available settings are:

- *Analytics Designer Properties*
 - *Name*
- *Quick Menus*
 - Configure the visibility of menu items at runtime
- *Size and Position*
- *Widget*
 - Background Color
 - Border
- *Actions*
 - *Show this item at view time*
 - *Always initialize on startup*
 - *Order*
- *Value Driver Tree Styling*
 - Minimum Node Width
- *Node Styling*
 - Header Background Color
- *Title Font*



Runtime Interactions

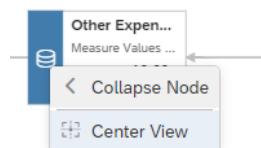
- Search Node



- Change Data



- Expand and Collapse Nodes



- Zoom and Pan

Script API

The Value Driver Tree widget provides, like other widgets, a script API such as:

```
// Returns the layout of the widget
getLayout(): Layout

// Returns whether the widget is visible
isVisible(): boolean

// Shows or hides the widget
setVisible(visible: boolean): void
```

5.30 Pause Refresh

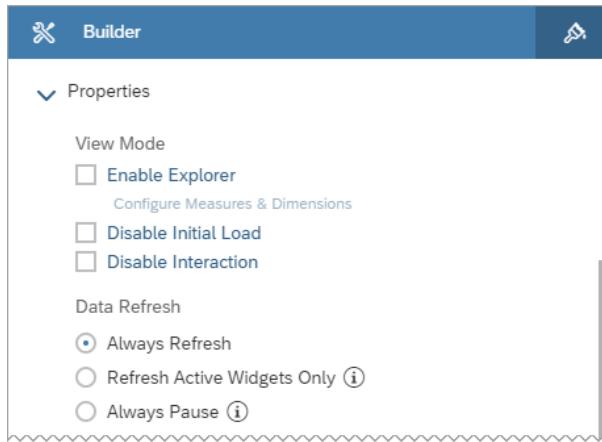
You, as an analytic application developer, can build applications, which allow analytic application end users to pause the refresh of a data source associated with a Table or Chart widget until they have completed their operations on that data source, such as, for example, adding or removing dimensions, filters, and key figures.

This optimizes runtime performance because an unnecessary initial data refresh is avoided, and remote queries won't be triggered each time after end users have taken an action.

When data refresh is resumed, then the widget is automatically refreshed, if necessary.

5.30.1 Builder Panel

To enable this capability at design time, you, as an analytic application developer, can select one of the *Data Refresh* options in the *Builder* panel of the Table or Chart widget.



The available options are the following:

- *Always Refresh* (default)
Data refresh is never paused.
- *Refresh Active Widgets Only*
For active widgets data refresh is never paused. For invisible widgets, data refresh is automatically paused (see detailed description below).
- *Always Pause*
Any operation resulting in a data refresh is paused.

5.30.1.1 Option Refresh Active Widgets Only

When you select the option *Refresh Active Widgets Only* for a Table or Chart widget, then the following rules apply to the widget:

- If the widget becomes visible, then its data refresh is resumed (“un-paused”).
- If the widget becomes invisible, then its data refresh is paused.
- If the widget is contained in an inactive popup dialog, then its data refresh is paused (such a widget is treated like an invisible widget).
- If the widget is contained in an inactive tab of a tab strip, then its data refresh is paused (such a widget is treated like an invisible widget).
- If the widget is accessed by a data-related script API method to fetch data, then its data refresh is resumed (“un-paused”). The specific script APIs methods are the following:
 - DataSource API applied to data sources associated with a Chart or Table widget
 - `getResultSet()`
 - `getData()`
 - `isResultEmpty()`
 - `getResultMember()`
 - `getDataSelections()`
 - `expandNode()`
 - `refreshData()`
 - Table API
 - `getRowCount()`
 - `getColumnCount()`
 - `getDataCells()`

- After the widget has been accessed by a data-related script API method to fetch data, then its data refresh is paused again.
- If a widget's data refresh is paused, then applying script API methods that change the result set of the data source associated with the widget (for example, `setDimensionFilter()` or `setVariableValue()`), trigger the execution of the `onResultChanged` event script.
- If the widget's visibility is changed several times in one script event, it's refreshed only once.

5.30.1.2 Disabling User Interaction

When the data refresh of a data source associated with a Table or Chart widget is paused, then end user interaction on such a widget becomes meaningless. Most of the end user's interaction will take no effect until data refresh is resumed. Thus, it's useful to enable or disable end user interactions on those widgets.

To enable or disable end user interaction at design time, deselect or select *Disable Interaction* in the *Builder* panel of the widget.

5.30.2 Script API

5.30.2.1 Single Widget

Besides configuring the Table or Chart widget in the *Builder* panel at design time, you, as an analytic application developer, can use the following script API to enable or disable data refresh at runtime as well:

```
Table.getDataSource().setRefreshPaused(paused: PauseMode | boolean);
Table.getDataSource().getRefreshPaused(): PauseMode;

Chart.getDataSource().setRefreshPaused(paused: PauseMode | boolean);
Chart.getDataSource().getRefreshPaused(): PauseMode;
```

with the following values for `PauseMode`:

```
PauseMode.On    // Always Pause
PauseMode.Off   // Always Refresh
PauseMode.Auto  // Refresh Active Widgets Only
```

Note:

- `setRefreshPaused(true)` is equivalent to `setRefreshPaused(PauseMode.On)`
- `setRefreshPaused(false)` is equivalent to `setRefreshPaused(PauseMode.Off)`

5.30.2.2 Multiple Widgets

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

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

5.30.2.3 Disabling User Interaction

When the data refresh of a data source associated with a Table or Chart widget is paused, then end user interaction on such a widget becomes meaningless. Most of the end user's interaction will take no effect until data refresh is resumed. Thus, it's useful to enable or disable end user interactions on those widgets.

You, as an analytic application developer, can use the following script API to enable or disable user interaction on the Table or Chart widget at runtime:

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

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

5.31 Data Explorer API

5.31.1 Showing or Hiding Additional Dimensions when Opening Explorer

When you open Explorer via script API, you, as an analytic application developer, can show or hide the dimensions added with `DataExplorer.setAdditionalDimensions()` with the following script API (by default, all additional dimensions are shown):

```
DataExplorer.setAdditionalDimensionsVisible(visible: boolean)
```

5.31.2 Configuring the Order of Additional Dimensions

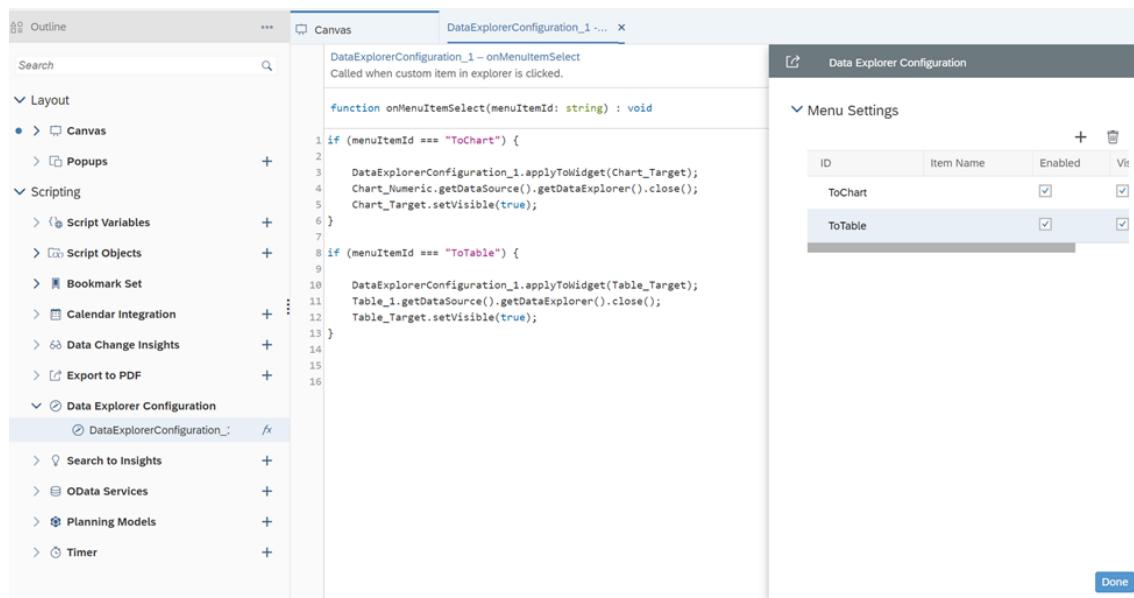
Some customers, for example, want to put the most important or most frequently used dimensions in the front. You, as an analytic application developer, can configure the order of the dimensions added with `DataExplorer.setAdditionalDimensions()` with the following script API:

```
DataExplorer.setAdditionalDimensionsSortOrder(sortOrder: DataExplorerSortOrder)

enum DataExplorerSortOrder {
    Argument, // Align with the order specified in setAdditionalDimensions()
    Default   // Sort in default order (ascending alphabetical order)
}
```

5.31.3 Applying Explorer Results to a Chart or Table

With the `DataExplorerConfiguration` object and its methods, you can apply Explorer results to a chart or table. Thus, your analytic application doesn't only offer pre-built charts and tables, but also provides business users the flexibility to do data exploration and apply the results to a chart or table and save those with bookmarks.



To apply Explorer results to a chart, add custom menu items to the Explorer's visualization dialog as follows:

1. In *Outline > Scripting > Data Explorer Configuration*, add a new configuration.
2. A *Data Explorer Configuration* panel opens at the right side of the Canvas.
3. In this panel, add custom menu items in the *Menu Settings*.
4. Add an `onMenuItemSelect` event script to the new configuration by clicking the fx button to the right of the new configuration in the *Outline*, then implement in the script the actions you want to trigger when these menu items are selected. The following example applies your changes to `Chart_1` in the Explorer dialog to `Chart_2`:

```
var explorer = Chart_1.getDataSource().getDataExplorer();
if (Chart_2.isVisible() === false) {
    DataExplorerConfiguration_1.applyToWidget(Chart_2);
    Chart_2.setVisible(true);
}
```

5. Save and run the analytic application.
6. Find your custom menu items in the Explorer.

Note: When you apply Explorer results to a chart or table, then the results of a chart can only be applied to a chart, and results of a table can only be applied to a table. The automatic transfer of the Show/Hide configuration in the Explorer, as well as the original chart or table styling to the target chart or table isn't supported.

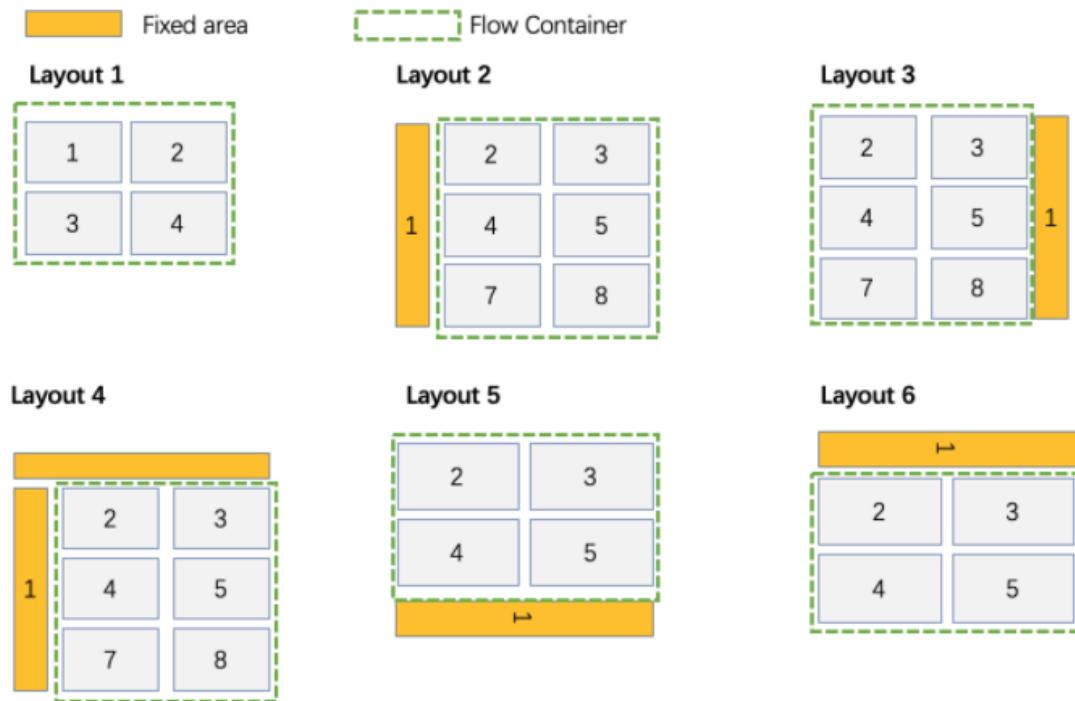
5.32 Flow Panel

The Flow Panel widget is a UI container. When the size of the Flow Panel widget changes, then widgets in the Flow Panel widget flow from one row to the next.

5.32.1 Fixed Area and Flow Container

You can use the Flow Panel widget and other widgets side by side to create typical layouts of analytic applications (see examples below).

For instance, in the example Layout 2, you want to have a sidebar always on the left and content, which will flow based on the size of the screen, always on the right. You can achieve this layout by using a normal Panel widget for the content on the left with its width set to 20% of the screen width, and a Flow Panel widget for the content on the right with its width set to 80% of the screen width.



Besides using a Flow Panel widget and other widgets side by side, you can even use a normal Panel widget nested inside a Flow Panel widget to create more sophisticated layouts.

5.32.2 Responsive Rule Configuration

The Flow Panel widget provides a user-friendly UI to configure the breakpoint rules for different screen sizes. It's like the media query concept used with Cascading Style Sheets (CSS). You can add a breakpoint where certain widgets' width and height will be adjusted differently at each breakpoint.

The screenshot shows the 'Responsive Rule Configuration' section of the 'Builder' interface. At the top, there is a toggle switch labeled 'Enable the rule in Design Time' with the value set to 'ON'. Below this, there are three separate configuration panels for different screen widths:

- Break Point 1:** Set for screen widths up to 600px. The 'Set the widget width' dropdown is set to 'Each Widget' with a value of 100%. There are two 'Add Widget' buttons.
- Break Point 2:** Set for screen widths up to 1500px. It includes two width settings: 'Each Widget' at 25% and 'O_Delete_Panel' at 100%. There are two 'Add Widget' buttons.
- Break Point 3:** Set for screen widths up to 1024px. It has one width setting for 'Each Widget' at 100%.

5.33 CSS Support

You can use Cascading Style Sheets (CSS) to style the widgets in your analytic application.

5.33.1 Supported Class Names and Properties

You can select a widget from the dropdown to view the supported CSS class names and properties.

Note: Not all CSS class names and properties are supported. With future releases more and more CSS class names and properties will be supported.

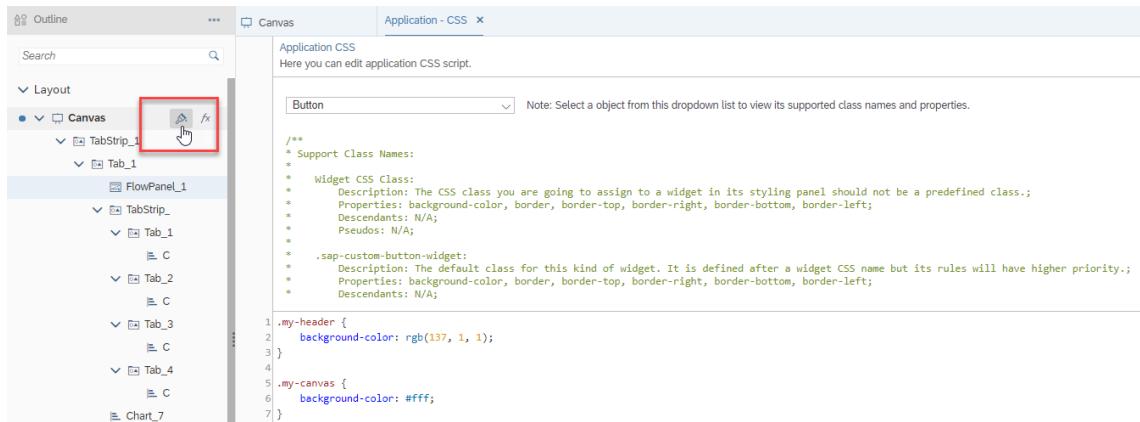


```
/*
* Support Class Names:
*
* Widget CSS Class:
*   Description: The CSS class you are going to assign to a widget in its styling panel should not be a predefined class.;
*   Properties: background-color, border, border-top, border-right, border-bottom, border-left;
*   Descendants: N/A;
*   Pseudos: N/A;
*
* .sap-custom-button-widget:
*   Description: The default class for this kind of widget. It is defined after a widget CSS name but its rules will have higher priority. ;
*   Properties: background-color, border, border-top, border-right, border-bottom, border-left;
*   Descendants: N/A;
*/
.my-header {
1  background-color: rgb(137, 1, 1);
2
3 }
4
5 .my-canvas {
6  background-color: #fff;
7 }
8
9 .my-main-tabstrip {
10  background-color: #fff;
11 }
12 }
```

5.33.2 Creating the Application CSS

Click on the *Edit CSS* icon besides the Canvas to open a new tab in which you can add the CSS classes that will be used later by each widget.

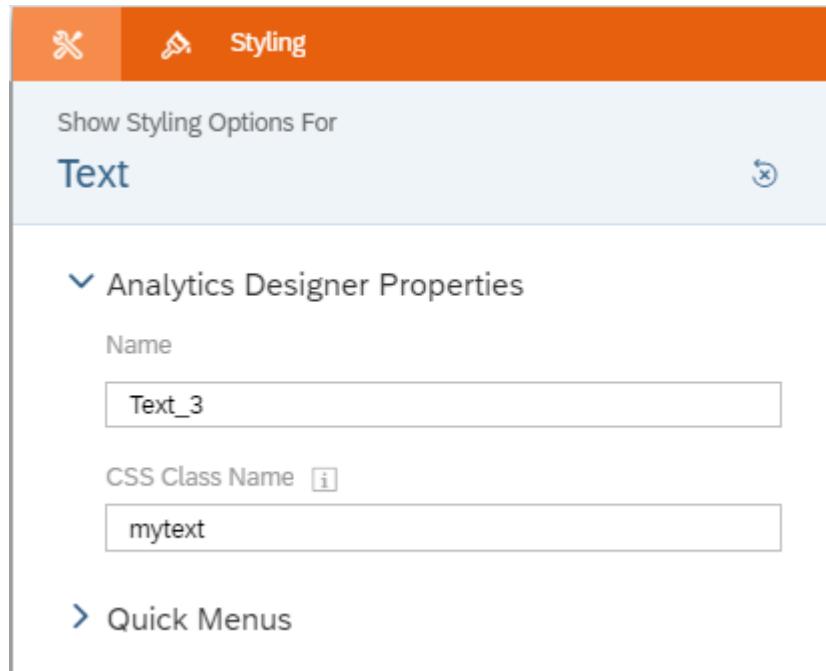
Saving this analytic application also saves this application CSS within the analytic application.



```
/*
* Support Class Names:
*
* Widget CSS Class:
*   Description: The CSS class you are going to assign to a widget in its styling panel should not be a predefined class.;
*   Properties: background-color, border, border-top, border-right, border-bottom, border-left;
*   Descendants: N/A;
*   Pseudos: N/A;
*
* .sap-custom-button-widget:
*   Description: The default class for this kind of widget. It is defined after a widget CSS name but its rules will have higher priority. ;
*   Properties: background-color, border, border-top, border-right, border-bottom, border-left;
*   Descendants: N/A;
*/
.my-header {
1  background-color: rgb(137, 1, 1);
2
3 }
4
5 .my-canvas {
6  background-color: #fff;
7 }
```

5.33.3 Using the CSS Class in a Widget

After you've created the application CSS, you can reference a CSS class in a widget. This CSS class has the highest priority over any CSS style setting on this widget.



5.33.4 Setting the Global Default Class Name

In addition to referencing a CSS class in a widget, you can also set a global default class name for all the related widgets in this analytic application without having to specify the class name for each widget.

For instance, the *Global Default Class Name* is set to `my-theme`, which includes the CSS settings for all Chart and Table widgets, so that you don't need to specify a CSS class for each specific Chart or Table widget.

The screenshot shows the SAP Fiori Launchpad settings interface. At the top, there are two tabs: 'Styling' (selected) and 'Script'. Below the tabs are two checkboxes: 'Snap to Grid' (checked) and 'Snap to Object' (checked). A section titled 'Canvas Size' has two radio buttons: 'Dynamic' (selected) and 'Fixed'. Under 'Canvas Content Layout', there are two radio buttons: 'Center Aligned' (unchecked) and 'Fit to Device' (selected). A checked checkbox labeled 'Device selection bar' is also present. A section titled 'Application Settings' contains a 'CSS Class Settings' heading. It includes a 'Canvas Class Name' input field with placeholder 'Enter class name' and a 'Global Default Class Name' input field with placeholder 'Enter the global default class name you define in Application.CSS and this class will be applied to all widgets, popups, and the canvas.' A note below states: 'Before generating PDF export, you need to first add an'. A code editor at the bottom displays the following CSS code:

```
/*
 * Charts
 */
.my-theme .sap-custom-chart-title,
.my-theme .sap-custom-chart-title {
    color: rgb(137, 1, 1);
}

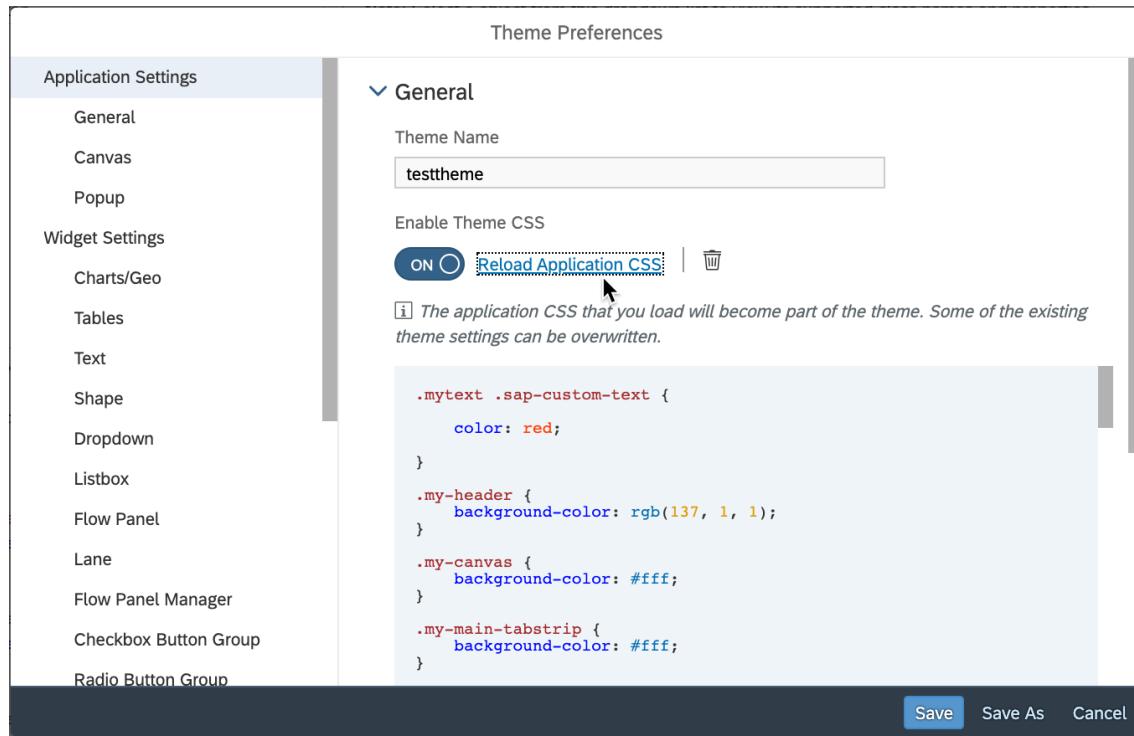
.my-theme .sap-custom-chart-subtitle,
.my-theme .sap-custom-chart-subtitle{
    color: rgba(137, 1, 1, 0.5);
}

.my-theme .sap-custom-chart-category-axis-label,
.my-theme .sap-custom-chart-legend,
.my-theme .sap-custom-chart-data-labels {
    fill: rgb(137, 1, 1);
}

/*
 * Table
 */
.my-theme .sap-custom-table-title,
.my-theme .sap-custom-table-dimension-member-expand-icon {
    color: rgb(137, 1, 1);
}
```

5.33.5 Loading CSS into a Theme

The CSS can be loaded into a theme and stored. The CSS that you've loaded becomes part of the theme. Some of the existing settings can be overwritten.

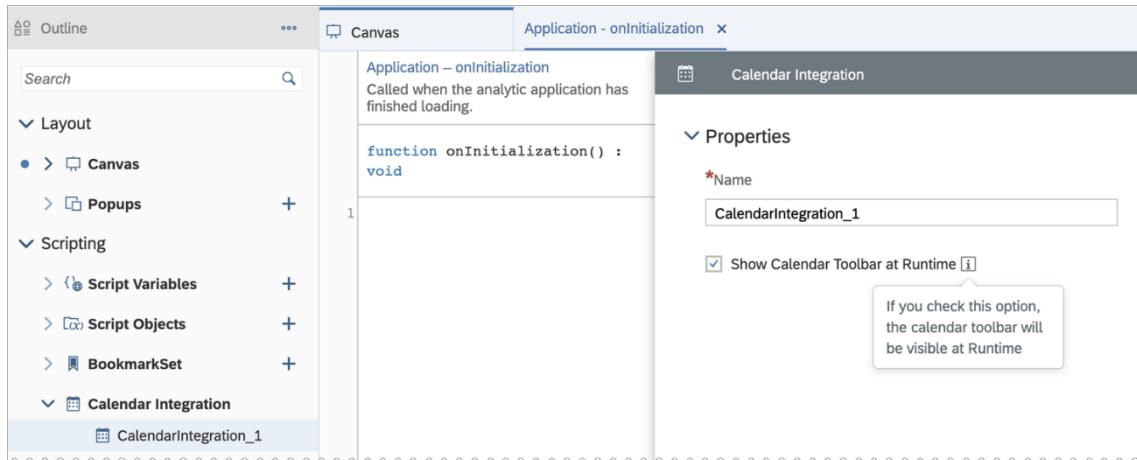


5.34 Calendar Task API

You can use the Calendar Task script API to submit, decline, approve, or reject a calendar task. The calendar task must be of type General Task, Review Task, or Composite Task. The script API is available on the Calendar Integration component.

5.34.1 Using the Calendar Integration Component

To create a Calendar Integration component, click in the *Scripting* section of the *Outline* the "+" (plus) icon next to *Calendar Integration*. The *Calendar Integration* panel opens where you can set the name of the Calendar Integration component. An additional checkbox *Show Calendar Toolbar at Runtime* allows you to control whether the specific calendar toolbar should be visible at runtime. By default, the calendar toolbar is shown.



You, as an analytic application developer, can use the following script API of the Calendar Integration component to retrieve the current calendar task, a `CalendarTask` object:

`CalendarIntegration.getCurrentTask(): CalendarTask`

You can use the following script API of the `CalendarTask` object to retrieve its status and type (for example, General Task, Composite Task, and Review Task) and you can also check if the currently logged-in user has a specific calendar task user role type in relation to this calendar task:

- `CalendarTask.getStatus(): CalendarTaskStatus`
- `CalendarTask.getType(): CalendarTaskType`
- `CalendarTask.hasUserRole(CalendarTaskUserRoleType): boolean`

Depending on the user role type, for example, Assignee, Owner, or Reviewer, you can use the following script API with the calendar task:

- `submit(): boolean`
- `decline(): boolean`
- `approve(): boolean`
- `reject(): boolean`

To use this script API, you first have to convert the generic calendar task, a `CalendarTask` object, to the target type of the calendar task, that is, a General Task, a Review Task, or a Composite Task (a `CalendarGeneralTask`, `CalendarReviewTask`, or `CalendarCompositeTask` object, respectively) using the `cast()` script API method.

Example:

With the following sample code snippet, you can cast a calendar task a Composite Task:

```
var calendarTask = CalendarIntegration_1.getCurrentTask();
if (calendarTask.getType() === CalendarTaskType.CompositeTask) {
    var compositeTask = cast(Type.CalendarCompositeTask, calendarTask);
    // ...
}
```

Note the check if the calendar task is of the correct target calendar task type before the cast operation takes place.

5.34.2 Using the getCurrentTask API

You can use the `getCurrentTask()` script API method to retrieve the calendar task that's loaded with your analytic application.

This is necessary because a user can't perform operations on a calendar task, for example, submit, decline, approve, or reject, if an analytic application isn't associated with a calendar task through a working file.

Example:

With the following sample code snippet, you can log the current task to the browser console:

```
var calendarTask = CalendarIntegration_1.getCurrentTask();
console.log(calendarTask);
```

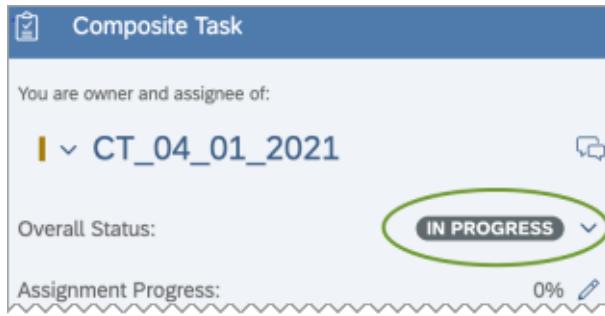
5.34.3 Using the getStatus API

You can examine the overall status of a calendar task with the `getStatus()` script API method. This script API returns a `CalendarTaskStatus` object. Note that `CalendarTaskStatus` represents only the most relevant subset of calendar task status that occur in calendar task workflows.

Example:

With the following sample code snippet, you can print the status of the calendar task displayed in the UI (see screenshot below) to the browser console:

```
var calendarTaskStatus = CalendarIntegration_1.getCurrentTask().getStatus();
switch (calendarTaskStatus) {
    case CalendarTaskStatus.Accomplished:
        console.log("Accomplished");
        break;
    case CalendarTaskStatus.InProgress:
        console.log("In Progress");
        break;
    case CalendarTaskStatus.Canceled:
        console.log("Canceled");
        break;
    case CalendarTaskStatus.OnHold:
        console.log("OnHold");
        break;
    case CalendarTaskStatus.Open:
        console.log("Open");
        break;
}
```



5.34.4 Using the `getType` API

You can retrieve the type of a calendar task with the `getType()` script API method to decide which Calendar Task script API to use to change the status of this calendar task. This script API returns the task type of the calendar task (General Task, Composite Task, or Review Task) as a `CalendarTaskType` object.

Example:

With the following sample code snippet, you can check if the current calendar task is a General Task:

```
var calendarTaskType = CalendarIntegration_1.getCurrentTask().getType();
if (calendarTaskType === CalendarTaskType.GeneralTask) {
    console.log("This is a general task.");
}
```

5.34.5 Using the `hasUserRole` API

You can examine if the currently logged-in user has a specific user role in relation to the calendar task with the `hasUserRole()` script API method. This script API accepts the following roles:

- `CalendarTaskUserRoleType.Reviewer`
- `CalendarTaskUserRoleType.Owner`
- `CalendarTaskUserRoleType.Assignee`

The script API returns `true` if the currently logged-in user has the specified user role, and `false` otherwise.

Example:

With the following sample code snippet, you can check if the currently logged-in user is a reviewer of the currently loaded calendar task:

```
var calendarTask = CalendarIntegration_1.getCurrentTask();
var isReviewer = calendarTask.hasUserRole(CalendarTaskUserRoleType.Reviewer);
console.log(isReviewer);
```

5.34.6 Using the `submit` API

As an assignee, you can use the `submit()` script API method to submit a General Task or a Composite Task. This script API returns `true` if the operation was successful, and `false` otherwise.

- `CalendarGeneralTask.submit(): boolean`

- `CalendarCompositeTask.submit(): boolean`

Example:

With the following sample code snippet, you can submit a Composite Task:

```
var calendarTask = CalendarIntegration_1.getCurrentTask();
var compositeTask = cast(Type.CalendarCompositeTask, calendarTask);
var isSubmitted = compositeTask.submit();
if (isSubmitted) {
    console.log("You've successfully submitted your task.");
} else {
    console.log("Sorry, something went wrong.");
}
```

This script API works like the *Submit* action on the side panel of the calendar.

5.34.7 Using the decline API

As an assignee, you can use the `decline()` script API method to decline a General Task or a Composite Task. This script API returns `true` if the operation was successful, and `false` otherwise.

- `CalendarGeneralTask.decline(): boolean`
- `CalendarCompositeTask.decline(): boolean`

Example:

With the following sample code snippet, you can decline a Composite Task:

```
var calendarTask = CalendarIntegration_1.getCurrentTask();
var compositeTask = cast(Type.CalendarCompositeTask, calendarTask);
var isDeclined = compositeTask.decline();
if (isDeclined) {
    console.log("You've successfully declined your task.");
} else {
    console.log("Sorry, something went wrong.");
}
```

This script API works like the *Decline* action on the side panel of the calendar.

5.34.8 Using the approve API

As a reviewer, you can use the `approve()` script API method to accept a Review Task or a Composite Task. Before this operation, the assignee needs to submit this task. This script API returns `true` if the operation was successful, and `false` otherwise.

- `CalendarReviewTask.approve(): boolean`
- `CalendarCompositeTask.approve(): boolean`

Example:

With the following sample code snippet, you can approve a Review Task:

```
var calendarTask = CalendarIntegration_1.getCurrentTask();
var reviewTask = cast(Type.CalendarReviewTask, calendarTask);
```

```
var isApproved = reviewTask.approve();
if (isApproved) {
    console.log("You've successfully approved the task.");
} else {
    console.log("Sorry, something went wrong.");
}
```

This script API works like the *Approve* action on the side panel of the calendar.

5.34.9 Using the reject API

As a reviewer, you can use the `reject()` script API method to reject a Review Task or a Composite Task to let the assignee have another look at this calendar task. Before this operation, the assignee needs to submit this task. This script API returns `true` if the operation was successful, and `false` otherwise.

- `CalendarReviewTask.reject(): boolean`
- `CalendarCompositeTask.reject(): boolean`

Example:

With the following sample code snippet, you can reject a Composite Task:

```
var calendarTask = CalendarIntegration_1.getCurrentTask();
var compositeTask = cast(Type.CalendarCompositeTask, calendarTask);
var isRejected = compositeTask.reject();
if (isRejected) {
    console.log("You've successfully rejected your task.");
} else {
    console.log("Sorry, something went wrong.");
}
```

This script API works like the *Reject* action on the side panel of the calendar.

5.34.10 Using the getCalendarTaskById API

You can use the `getCalendarTaskById()` script API method to retrieve information about a specific task. Simply pass the event ID from the query string in the URL. The analytic application doesn't need to be opened by clicking a work file from the calendar task to use this API.

Example:

With the following sample code snippet, you can log the result of the `getType()` script API method to the browser console for any task:

```
var task = CalendarIntegration_1.getCalendarTaskById("taskId");
if (task.getType() === CalendarTaskType.CompositeTask) {
    console.log("This is a composite task");
} else if (task.getType() === CalendarTaskType.GeneralTask) {
    console.log("This is a general task");
} else if (task.getType() === CalendarTaskType.ReviewTask) {
    console.log("This is a review task");
}
```

5.34.11 Using the activate API

You can use the `activate()` script API method to activate a task and optionally specify to issue a notification to all users that have a role on the task.

Example:

With the following sample code snippet, you can activate a task and issue a notification:

```
var task = CalendarIntegration_1.getCalendarTaskById("taskId");
task.activate(true);
```

This script API works like the *Activate* or *Activate and Notify* action from the calendar view.

5.34.12 Using the getRelatedTasksId API

You can use the `getRelatedTaskIds()` script API method to retrieve all the calendar task IDs for which the opened analytic application is a working file. The API returns either an array of task IDs or, if no tasks are found, an empty array.

Note: Recurring composite tasks are currently not delivered as part of the result.

Example:

With the following sample code snippet, you can retrieve all tasks for which the opened analytic application is a working file:

```
var allTasks = CalendarIntegration_1.getRelatedTaskIds();
console.log(allTasks);
```

5.34.13 Using the createCompositeTask API

You can use the `createCompositeTask()` script API method to create a composite task with properties and options.

When creating a composite task, you can choose a name, a start date, and a due date, which are mandatory properties. You can choose between an analytic application or a story when specifying the work file.

Example:

With the following sample code snippet, you can create a Composite Task is created that starts now and ends on March 16, 2022. There is one work file belonging to this composite task, the currently opened application. One assignee is defined, the currently logged-in user. Additionally, a reviewer is specified, in one review round:

```
var newTask = CalendarIntegration_1.createCompositeTask({
  name: "newCompositeTask",
  startDate: new Date(Date.now()),
  dueDate: new Date(2022, 2, 16, 0, 0, 0),
  workFiles: [
    {
      id: Application.getInfo().id,
      type: CalendarTaskWorkFileType.AnalyticApplication,
    },
    assignees: [Application.getUserInfo().id],
    description: "Calendar Composite Task description",
  ],
});
```

```
reviewers: {
    "1": ["reviewerId"],
}
}, {
    autoActivate: true
});
```

The option `autoActivate` replaces the *Activate the task at start date automatically* option in the user interface.

5.34.14 Using the Status Change APIs for any Calendar Composite Task

You can use the `submit()`, `decline()`, `approve()`, or `reject()` script API methods for any composite task, given the task ID.

Example:

With the following sample code snippet, you can submit a Composite Task with a given task ID:

```
var task = CalendarIntegration_1.getCalendarTaskById("taskId");
if (task.getType() === CalendarTaskType.CompositeTask) {
    var compositeTask = cast(Type.CalendarCompositeTask, task);
    compositeTask.submit();
}
```

5.34.15 Using the canUserSubmit API

You can use the `canUserSubmit()` script API method to check if a calendar task is ready to be submitted. If you use this method when the task status is *Open*, then the task status is changed to *In Progress*.

Example:

With the following sample code snippet, you can submit a Composite Task with the ID "`myTaskId`" that has the task status *Open*:

```
var task = CalendarIntegration_1.getCalendarTaskById("myTaskId");
var compositeTask = cast(Type.CalendarCompositeTask, task);
var canSubmit = compositeTask.canUserSubmit();
if (canSubmit) {
    var isSubmitted = compositeTask.submit();
    if (isSubmitted) {
        console.log("You've successfully submitted your task.");
    } else {
        console.log("Something went wrong when submitting your task.");
    }
} else {
    console.log("Sorry, you can't submit this composite task.");
}
```

5.34.16 Using the canUserApprove API

You can use the `canUserApprove()` script API method for the user role *Reviewer* on a Composite Task or for the user role *Assignee* on a Review Task to verify if the task can be approved.

Example:

With the following sample code snippet, you can approve all the related Review Tasks and Composite Tasks that can be approved by the currently logged-in user:

```
var allTasks = CalendarIntegration_1.getRelatedTaskIds();
for (var i = 0; i < allTasks.length; i++) {
    var task = CalendarIntegration_1.getCalendarTaskById(allTasks[i]);
    if (task.getType() === CalendarTaskType.CompositeTask) {
        var compositeTask = cast(Type.CalendarCompositeTask, task);
        var canApprove = compositeTask.canUserApprove();
        if (canApprove) {
            console.log("Approving composite task with name " + task.getName() + " and ID " + task.getId() + "...");
            var isApprovedCompositeTask = compositeTask.approve();
            if (isApprovedCompositeTask) {
                console.log("You've successfully approved this composite task.");
            } else {
                console.log("Something went wrong when approving this composite task.");
            }
        } else {
            console.log("Sorry, you can't approve this composite task.");
        }
    } else if (task.getType() === CalendarTaskType.ReviewTask){
        var reviewTask = cast(Type.CalendarReviewTask, task);
        var canApproveReviewTask= reviewTask.canUserApprove();
        if (canApproveReviewTask) {
            console.log("Approving review task with name " + task.getName() + " and ID " + task.getId() + "...");
            var isApprovedReviewTask = reviewTask.approve();
            if (isApprovedReviewTask) {
                console.log("You've successfully approved this review task.");
            } else {
                console.log("Something went wrong when approving this review task.");
            }
        } else {
            console.log("Sorry, you can't approve this review task.");
        }
    }
}
```

5.34.17 Using the canUserReject API

You can use the `canUserReject()` script API method for the user role *Reviewer* on a Composite Task or for the user role *Assignee* on a Review Task to check if the task can be rejected.

Example:

With the following sample code snippet, you can reject all Composite Tasks for the currently logged-in user:

```
var allTasks = CalendarIntegration_1.getRelatedTaskIds();
for (var i = 0; i < allTasks.length; i++) {
    var task = CalendarIntegration_1.getCalendarTaskById(allTasks[i]);
    if (task.getType() === CalendarTaskType.CompositeTask) {
        var compositeTask = cast (Type.CalendarCompositeTask, task);
        var canReject = compositeTask.canUserReject();
        if (canReject) {
            console.log("Rejecting task with name " + task.getName() + " and ID " +
task.getId());
            var isRejectedCompositeTask = compositeTask.reject();
            if (isRejectedCompositeTask) {
                console.log("You've successfully rejected this composite task.");
            } else {
                console.log("Something went wrong when rejecting this composite
task.");
            }
        } else {
            console.log("Sorry, you can't reject this composite task.");
        }
    }
}
```

5.34.18 Using the canUserDecline API

You can use the [canUserDecline\(\)](#) script API method for the user role *Assignee* on a Composite Task or a General Task to verify if the task can be declined.

Example:

With the following sample code snippet, you can verify if the currently logged-in user can decline the current task:

```
var task = CalendarIntegration_1.getCurrentTask();
if (task.getType() === CalendarTaskType.CompositeTask) {
    var compositeTask= cast(Type.CalendarCompositeTask, task);
    var canDeclineCompositeTask = compositeTask.canUserDecline();
    if (canDeclineCompositeTask) {
        console.log("You can decline this composite task.");
    } else {
        console.log("You can't decline this composite task.");
    }
} else if (task.getType() === CalendarTaskType.GeneralTask){
    var generalTask = cast (Type.CalendarGeneralTask, task);
    var canDeclineGeneralTask = generalTask.canUserDecline();
    if (canDeclineGeneralTask) {
        console.log("You can decline this general task.");
    } else {
        console.log("You can't decline this general task.");
```

```
    }  
}
```

5.35 Data Actions

In addition to the Data Action Trigger widget, you can also use the Data Action component and its related script API to run short-running data actions as well as set and read parameter values.

Data actions are a flexible planning tool for making structured changes to model data, including copying data from one model to another. In analytics designer there are two ways of adding data actions to your application:

- Using the Data Action Trigger widget and its related script API
- Using the Data Action component and its related script API

With the Data Action component and its related script API you can do the following:

- Execute the data action
- Set the value of a data action parameter
- Read the value of a data action parameter

5.35.1 Prerequisites

Before you can use a Data Action component, there must be data actions created already.

To work with the Data Action component, you need the following permissions and roles:

- To create or edit a Data Action component, you need the permission to edit the analytic application.
- To select a data action in the *Data Action Configuration* panel you need the respective read permission for data actions.
- To execute a data action, you need the respective execute permission for the selected data action.

5.35.2 Adding a Data Action Component

You need to add a Data Action component to the analytic application to be able to use the related script API.

1. To add a Data Action component, click in the *Scripting* section of the *Outline* the “+” (plus) icon next to *Data Actions*. The *Data Action Configuration* panel opens with an empty description field where you can type in a description for the data action.
2. In the *Data Action* dropdown select one of the data actions that have been created already. The related data action parameters are displayed below.
3. Enter the values for the parameters either by using the member selection or by choosing the given default value that has been set in the data action designer. Note that to execute a data action successfully, all parameters have to be filled with values, either in the *Data Action Configuration* panel or with the `setParameterValue()` script API method. Note also that the design of the data action itself can't be changed in the *Data Action Configuration* panel, only the parameter values can be set here to adjust the data action.
4. Click *Done* when you've finished setting the parameters.

5.35.3 Script API

After adding the Data Action component, you, as an analytic application developer, can use a script API to work with data actions in your analytic application to set or read a parameter value as well as to execute data actions.

5.35.3.1 Setting a Parameter Value of a Data Action

The following script API method sets a parameter of a data action:

```
DataAction.setParameterValue(id: string, value: string | string[] | DataActionParameterValue | number): void
```

Example:

In the following example, a set of dimension member filters of dimension `dimensionId` is assigned to parameter `parameterId` of a data action. The code collects the relevant single value and multiple value dimension member filters into an array before assigning them to the parameter. Exclusive filters are ignored, as they aren't supported by the data action.

```
var filters = Table_1.getDataSource().getDimensionFilters(dimensionId);
var filteredMemberIds = ArrayUtils.create(Type.string);

for (var i = 0; i < filters.length; i++) {
    if (filters[i].type === FilterValueType.Single) {
        var singleFilter = cast(Type.SingleFilterValue, filters[i]);
        if (!singleFilter.exclude) {
            filteredMemberIds.push(singleFilter.value);
        }
    } else if (filters[i].type === FilterValueType.Multiple) {
        var multiFilter = cast(Type.MultipleFilterValue, filters[i]);
        if (!multiFilter.exclude) {
            filteredMemberIds = filteredMemberIds.concat(multiFilter.values);
        }
    }
}

DataAction_1.setParameterValue(parameterId, {
    type: DataActionParameterValueType.Member,
    members: filteredMemberIds
});
```

5.35.3.2 Reading a Parameter Value of a Data Action

The following script API method reads a parameter of a data action:

```
DataAction.getParameterValue(id: string): DataActionParameterValue
```

Example:

In the following example, the value of parameter `parameterId` of a data action is printed to the browser console. The value can be a number or an array of dimension member IDs.

```
var parameterValue = DataAction_1.getParameterValue(parameterId);
if (parameterValue.type === DataActionParameterValueType.Number) {
    var numberParameter = cast(Type.DataActionNumberParameterValue, parameterValue);
```

```

        console.log(numberParameter.value);
    } else { // parameterValue.type === DataActionParameterValueType.Member
        var memberParameter = cast(Type.DataActionMemberParameterValue, parameterValue);
        var memberIds = memberParameter.members;
        for (var i = 0; i < memberIds.length; i++) {
            console.log(memberIds[i]);
        }
    }
}

```

5.35.3.3 Executing a Data Action

The following script API method executes a data action:

```
DataAction.execute(): DataActionExecutionResponse
```

Note: The data action is executed as a blocking operation. A blocking operation prevents the rest of the application script from running until the data action is complete. It's best to use it only for data actions that take a short time to run.

Example:

In the following example, a data action is run, and a message is printed to the browser console whether the operation was successful or not.

```

var response = DataAction_1.execute();
if (response.status === DataActionExecutionResponseStatus.Success) {
    console.log("The execution of your data action was successful.");
} else { // response.status === DataActionExecutionResponseStatus.Error
    console.log("Sorry, the execution of your data action failed.");
}

```

5.36 Comments

5.36.1 Working with Comments

Besides end users that directly create or remove comments in an analytic application as in a story, you, as an analytic application developer, can add, view, delete comments, and so on, via a script API (available for commenting by data point in planning models only).

5.36.1.1 Script API

Managing Comments

You, as an analytic application developer, can manage data cell-based comments with a script API and specifying the comment ID or a data context (selection).

Example:

In the following example, a comment is added to a table cell:

```

Table_1.getDataSource().getComments().addComment({@MeasureDimension:
    "[sap.epm:M010_10_Accounts].[parentId].&[A134000]",
    sap.epm:M010_10_Operating_Income_Version: "public.Actual"}, "comment1");

```

Example:

In the following example, a comment is removed using the comment ID:

```
Table_1.getDataSource().getComments().removeComment("28760729-2540-4227-b016-428450515042");
```

Example:

In the following example, all comments (specified by a selection) are removed:

```
Table_1.getDataSource().getComments().removeAllComments({@MeasureDimension: "[sap.epm:M010_10_Accounts].[parentId].&[A134000]", sap.epm:M010_10_Operating_Income_Version: "public.Actual"});
```

Retrieving Comments

Besides end users posting or removing comments, you, as an analytic application developer, can read the comment information by comment ID or data context (selection) with a script API. The comment information includes content, author, creation date, and so on.

Example:

In the following example, a comment is retrieved using the comment ID:

```
var oCommentInfo2 = Table_1.getDataSource().getComments().getComment("28760729-2540-4227-b016-428450515042");
```

In the following example, all comments (specified by a selection) are retrieved:

```
var aCommentInfos =
Table_1.getDataSource().getComments().getAllComments({@MeasureDimension: "[sap.epm:M010_10_Accounts].[parentId].&[A134000]", sap.epm:M010_10_Operating_Income_Version: "public.Actual"});
```

Turning On and Off Comment Display

You, as an analytic application developer, can turn on and off the display of comments via a script API. Once the comment mode is disabled, the comments and related UI are invisible at runtime.

```
Application.isCommentModeEnabled();
Application.setCommentModeEnabled(true);
```

Liking a Comment

You, as an analytic application developer, can like a comment with a script API. Once a comment ID is specified, the number of likes of the related comment is updated.

```
Table_1.getDataSource().getComments().setCommentLiked(("28760729-2540-4227-b016-428450515042", true);
```

5.36.2 Comment Widget

The Comment widget helps you to synchronize the comments of the same data cell in a table. It also helps end users to manipulate table data cell comments in the same context.

5.36.2.1 Creating a Comment Widget

To add a Comment widget, in analytics designer click the "+" (plus) icon in the toolbar and select *Comment Widget* from the menu.

5.36.2.2 Configuring a Comment Widget

You can configure the data source and filters in the *Builder* panel to specify the data cells relevant to commenting:

The screenshot shows the 'Builder' panel with the following configuration:

- Data Source:** BestRunJuice_SampleModel
- Filters:**
 - Category (1)**: Actual
 - Account (1)**: Gross Margin
 - Location (1)**: California
- Add Filters** button

You can configure the styling options in the *Styling* panel:

The screenshot shows the 'Styling' panel for a 'Comment Widget' with the following configuration:

- Analytics Designer Properties** section:
 - Name: CommentWidget_2
 - CSS Class Name: Enter a class name (overwrites global default class)
- Quick Menus**
- Size and Position** section:

Width	384 px	Height	384 px
Left (X)	416 px	Top (Y)	416 px
Right	auto	Bottom	auto
- Widget** section:
 - Background Color: (dropdown menu)
 - Border: No Border
- Actions** section:
 - Show this item at view time:
 - Always initialize on startup:
 - Order: (four icons for reordering)

Among the available options are the following:

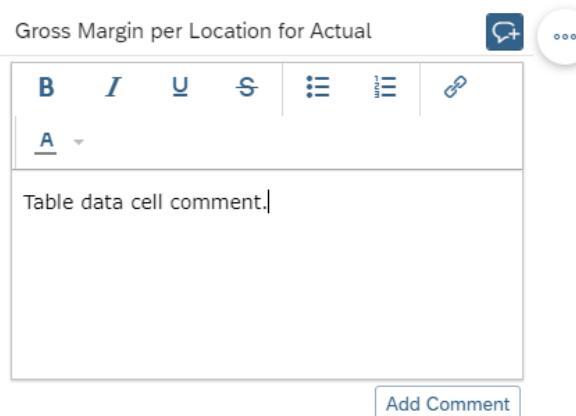
- *Analytics Designer Properties*
 - *Name*
- *Quick Menus*
- *Size and Position*
- *Widget*
 - *Background Color*
 - *Border*

- *Actions*
 - *Show this item at view time*
 - *Always initialize on startup*
 - *Order*

5.36.2.3 Using a Comment Widget

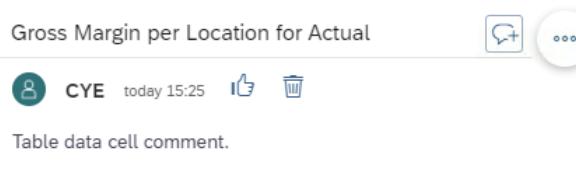
Adding a Comment

At runtime, end users can add comments in the following dialog:



Liking and Deleting a Comment

At runtime, end users can like or delete a comment by clicking the respective icon at the top of each comment:



5.36.2.4 Script API

The Comment widget provides a script API that's common to all widgets:

Returns the layout of the widget:

```
Commenting.getLayout(): Layout
```

Returns whether the widget is visible:

```
Commenting.isVisible(): boolean
```

Shows or hides the widget:

```
Commenting.setVisible(visible: boolean): void
```

Sets the Cascading Styling Sheet (CSS) class name of the component:

```
Commenting.setCssClass(className: string): void
```

Returns the Cascading Styling Sheet (CSS) class name of the component:

```
Commenting.getCssClass(): string
```

In addition, the Comment widget provides a data source-related script API:

Returns the data source of the Comment widget:

```
Commenting.getCommentingDataSource(): CommentingDataSource
```

where the class `CommentingDataSource` provides the following script API:

Sets the dimension filters of a Comment widget:

```
CommentingDataSource.setDimensionFilter(dimension: string | DimensionInfo, member: string | string[] | MemberInfo | MemberInfo[] | MeasureInfo | MeasureInfo[] | TimeRange | TimeRange[])
```

Returns the dimension filters of a Comment widget:

```
CommentingDataSource.getDimensionFilters(dimension: string | DimensionInfo) : FilterValue[]
```

Removes the dimension filters of a Comment widget:

```
CommentingDataSource.removeDimensionFilter(dimension: string | DimensionInfo)
```

Example:

In the following example, the dimension filter of dimension *Location* from a table is copied to a Comment widget:

```
var filter = Table_1.getDataSource().getDimensionFilters("Location")[0];
if (filter.type === FilterValueType.Single) {
    // Table_1 has a single filter, for example, "location=SA1"
    var singleFilter = cast(Type.SingleFilterValue, filter);
    CommentWidget_1.getCommentingDataSource().setDimensionFilter("Location",
        singleFilter.value);
} else if (filter.type === FilterValueType.Multiple) {
    // Table_1 has multiple filters, for example, "location=SA1" and "location=SA2"
    var multipleFilter = cast(Type.MultipleFilterValue, filter);
    CommentWidget_1.getCommentingDataSource().setDimensionFilter("Location",
        multipleFilter.values);
}
```

5.37 Export API

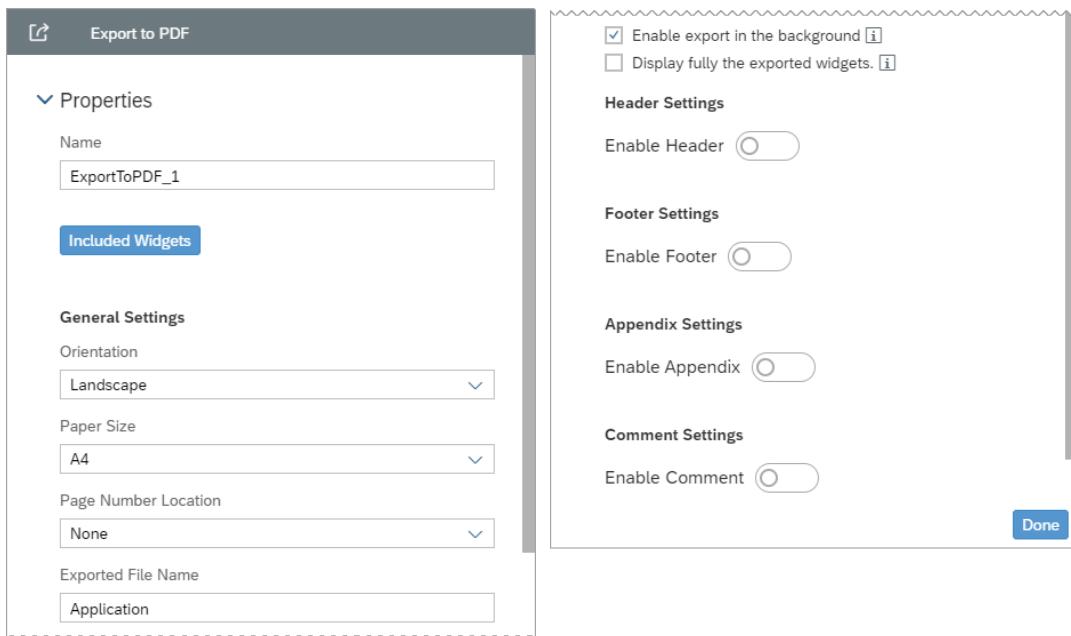
End users can select *Export* from the context menu of a table or a chart to export analytic application content in several output formats. In addition, you, as an analytic application developer, can use export components and their script APIs to export analytic application content, especially tables.

5.37.1 Export to PDF

You, as an analytic application developer, can use the Export to PDF component and its script API to export application content and especially table content to a PDF file.

5.37.1.1 Configuration

To add an Export to PDF component, click in the *Scripting* section of the *Outline* the “+” (plus) icon next to *Export to PDF*. This creates the Export to PDF component and opens its *Export to PDF* panel:



This panel lets you configure the following properties:

- **Name**
You can specify the component name.
- **Included Widgets**
You can select which widgets are exported to the PDF file. Note that it doesn't matter whether the widgets are shown or hidden in the application at runtime.
- **Orientation**
You can specify the paper orientation of the PDF file, *Landscape* or *Portrait*.
- **Paper Size**
You can specify the paper size of the of the PDF file, for example, *Letter*, *A4*, or *Auto*.
- **Paper Number Location**
You can specify where the PDF file pages display the page number, in the header or footer or don't display a page number at all.
- **Exported File Name**
You can specify the file name of the exported file. *Application* is the default file name.
- **Enable export in the background**
You can specify whether the file is exported in the background.
- **Display fully the exported widgets**
You can specify whether tables are exported in full length (report) when the analytic application is exported to a PDF file with the script API method `exportView()`.
- **Enable Header**
You can specify whether the PDF file pages contains a header.
- **Enable Footer**
You can specify whether the PDF file pages contains a footer.
- **Enable Appendix**
You can specify whether the PDF file contains an appendix.
- **Enable Comment**
You can specify whether the PDF file contains comments.

5.37.1.2 Script API

The following script API lets you configure and export application content to a PDF file:

Includes a widget or a popup in the exported PDF file:

```
ExportPdf.includeComponent(component: Widget | Popup): void
```

Excludes a widget or a popup from the exported PDF file:

```
ExportPdf.excludeComponent(component: Widget | Popup): void
```

Sets the page orientation of the exported PDF file:

```
ExportPdf.setPageOrientation(orientation: PageOrientation): void
```

Returns the page orientation of the exported PDF file:

```
ExportPdf.getPageOrientation(): PageOrientation
```

Sets the page size of the exported PDF file:

```
ExportPdf.setPageSize(size: PageSize): void
```

Returns the page size of the exported PDF file:

```
ExportPdf.getPageSize(): PageSize
```

Sets the location of the page number in the exported PDF file:

```
ExportPdf.setPageNumberLocation(location: PageNumberLocation): void
```

Returns the location of the page number in the exported PDF file:

```
ExportPdf.getPageNumberLocation(): PageNumberLocation
```

Sets the filename of the exported PDF file:

```
ExportPdf.setFileName(fileName: string): void
```

Returns the filename of the exported PDF file:

```
ExportPdf.getFileName(): string
```

Enables or disables the export of the PDF file in the background:

```
ExportPdf.setExportInBackgroundEnabled(isEnabled: boolean): void
```

Returns whether the export of the PDF file in the background is enabled:

```
ExportPdf.isExportInBackgroundEnabled(): boolean
```

Includes or excludes the export of all included tables in full length (report) when the analytic application is exported to a PDF file with `exportView()`:

```
ExportPdf.setReportIncluded(included: boolean): void
```

Returns whether the export of all included tables in full length (report) is enabled when the analytic application is exported to a PDF file with `exportView()`:

```
ExportPdf.isReportIncluded(): boolean
```

Shows or hides the page header in the exported PDF file:

```
ExportPdf.setHeaderVisible(visible: boolean): void
```

Returns whether the page header is visible in the exported PDF file:

```
ExportPdf.isHeaderVisible(): boolean
```

Sets the text shown in the page header of the exported PDF file:

```
ExportPdf.setHeaderText(text: string): void
```

Returns the text shown in the page header of the exported PDF file:

```
ExportPdf.getHeaderText(): string
```

Shows or hides the page footer in the exported PDF file:

```
ExportPdf.setFooterVisible(visible: boolean): void
```

Returns whether the page footer is visible in the exported PDF file:

```
ExportPdf.isFooterVisible(): boolean
```

Sets the text shown in the page footer of the exported PDF file:

```
ExportPdf.setFooterText(text: string): void
```

Returns the text shown in the page footer of the exported PDF file:

```
ExportPdf.getFooterText(): string
```

Shows or hides the appendix in the exported PDF file:

```
ExportPdf.setAppendixVisible(visible: boolean): void
```

Returns whether the appendix is visible in the exported PDF file:

```
ExportPdf.isAppendixVisible(): boolean
```

Shows or hides the comments in the exported PDF file:

```
ExportPdf.setCommentsVisible(visible: boolean): void
```

Returns whether the comments are visible in the exported PDF file:

```
ExportPdf.isCommentsVisible(): boolean
```

Exports the analytic application to a PDF file. Note that you can specify which widgets of the analytic application are exported in the *Included Widgets* dialog. You can open this dialog by clicking *Included Widgets* in the *Export to PDF* panel.

```
ExportPdf.exportView(): boolean
```

Exports tables in full length (report) to a PDF file. Note that you can specify which tables are exported in the *Included Widgets* dialog. You can open this dialog by clicking *Included Widgets* in the *Export to PDF* panel.

```
ExportPdf.exportReport(): void
```

Example:

In the following example, the analytic application is exported to PDF file *Export1.pdf*:

```
ExportToPDF_1.setFileName("Export1.pdf");
ExportToPDF_1.exportView();
```

Example:

In the following example, all tables are exported in full length (report) to file *Export2.pdf* (assuming all tables have been included in the *Included Widgets* dialog):

```
ExportToPDF_1.setFileName("Export2.pdf");
ExportToPDF_1.exportReport();
```

Example:

In the following example, tables *Table_1* and *Table_2* are exported in full length (report) to file *Export3.pdf* (assuming no other tables have been included in the *Included Widgets* dialog):

```
ExportToPDF_1.setFileName("Export3.pdf");
ExportToPDF_1.includeComponent(Table_1);
ExportToPDF_1.includeComponent(Table_2);
ExportToPDF_1.exportReport();
```

Example:

In the following example, the application is exported to PDF file *Export4.pdf*. Tables [Table_1](#) and [Table_2](#) are appended in full length (report) to the exported file.

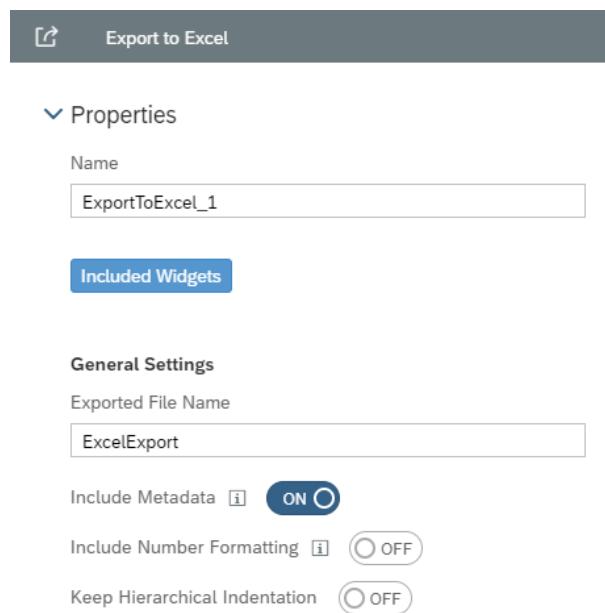
```
ExportToPDF_1.setFileName("Export4.pdf");
ExportToPDF_1.setReportIncluded(true);
ExportToPDF_1.includeComponent(Table_1);
ExportToPDF_1.includeComponent(Table_2);
ExportToPDF_1.exportView();
```

5.37.2 Export to Excel

You, as an analytic application developer, can use the Export to Excel component and its script API to export table content to an Excel file.

5.37.2.1 Configuration

To add an Export to Excel component, click in the *Scripting* section of the *Outline* the “+” (plus) icon next to *Export to Excel*. This creates the Export to Excel component and opens its *Export to Excel* panel:



This panel lets you configure the following properties:

- *Name*

You can specify the component name.

- *Included Widgets*

You can select which widgets are exported to the Excel file. Currently only tables are supported. Tables in popups and container widgets are available for export as well. Each table is placed on a separate Excel sheet.

- *Exported File Name*

You can specify the file name of the exported file. *Application* is the default file name.

- *Include Metadata*

You can specify whether the table metadata is exported as well. Table metadata is placed on the last Excel sheet.

- *Include Number Formatting*

You can specify whether the data values are exported with the scaling, units, and currencies as defined in the model and application.

- *Keep Hierarchical Indentation*

You can specify whether to keep the hierarchical indentation of the data labels.

5.37.2.2 Script API

The following script API lets you configure and export table content to an Excel file. Each table is placed on a separate Excel sheet.

```
ExportExcel.exportReport(): void  
ExportExcel.setFileName(text: string): void  
ExportExcel.getFileName(): string  
ExportExcel.setWidget(table: Table | Table[]): void  
ExportExcel.setExportFormattedValues(exportFormattedValues: boolean): void  
ExportExcel.isExportFormattedValues(): boolean  
ExportExcel.setIndentedHierarchy(indentedHierarchy: boolean): void  
ExportExcel.isIndentedHierarchy(): boolean  
ExportExcel.setAppendixIncluded(included: boolean): void  
ExportExcel.isAppendixIncluded(): boolean
```

Example:

In the following example, tables `Table_1` and `Table_2` are exported to Excel file `Export.xlsx`, including their metadata. In the exported Excel file, `Table_1` is placed on the first sheet, `Table_2` on the second sheet, and the metadata on the third sheet.

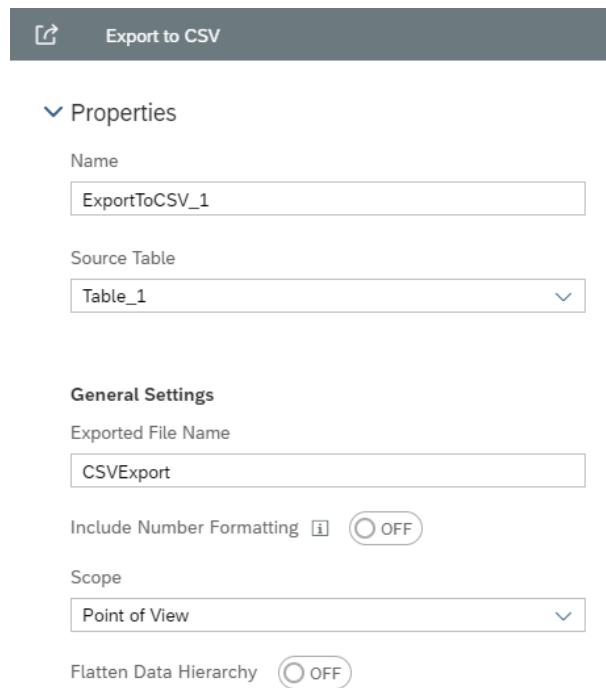
```
ExportToExcel_1.setFileName("Export.xlsx");  
ExportToExcel_1.setWidget([Table_1, Table_2]);  
ExportToExcel_1.setAppendixIncluded(true);  
ExportToExcel_1.setExportFormattedValues(true);  
ExportToExcel_1.setIndentedHierarchy(true);  
ExportToExcel_1.exportReport();
```

5.37.3 Export to CSV

You, as an analytic application developer, can use the Export to CSV component and its script API to export table content to a CSV (Comma Separated Values) file.

5.37.3.1 Configuration

To add an Export to CSV component, click in the *Scripting* section of the *Outline* the “+” (plus) icon next to *Export to CSV*. This creates the Export to CSV component and opens its *Export to CSV* panel:



This panel lets you configure the following properties:

- *Component Name*
You can specify the component name.
- *Source Table*
You can select which table is exported to a CSV file.
- *Exported File Name*
You can specify the file name of the exported file. *Application* is the default name.
- *Include Number Formatting*
You can specify whether the data values are exported with the scaling, units, and currencies as defined in the model and application.
- *Scope*
You can specify one of the following scopes, which define what data is exported:
 - *Point of View* – Exports the table data that's currently displayed, for example, only the data of expanded hierarchy nodes (default).
 - *All* – Export all the table data, for example, data of both expanded and collapsed hierarchy nodes.
- *Flatten Data Hierarchy*
You can specify whether separate hierarchy levels are exported to individual columns. This option is only available when *Scope* is set to *Point of View*. When *Scope* is set to *All*, then separate hierarchy levels are exported into a single column.

5.37.3.2 Script API

The following script API lets you configure and export table content to an CSV file:

```
ExportCsv.exportReport(): void
ExportCsv.setFileName(text: string): void
ExportCsv.getFileName(): string
ExportCsv.setWidget(table: Table): void
```

```

ExportCsv.setExportFormattedValues(exportFormattedValues: boolean): void
ExportCsv.isExportFormattedValues(): boolean
ExportCsv.setHierarchyLevelsInIndividualCells(separate: boolean): void
ExportCsv.isHierarchyLevelsInIndividualCells(): boolean
ExportCsv.setScope(scope: ExportScope): void
ExportCsv.getScope(): ExportScope

```

Example:

In the following example, table `Table_1` is completely exported to CSV file `Export.csv`:

```

ExportToCsv_1.setFileName("Export.csv");
ExportToCsv_1.setWidget(Table_1);
ExportToCsv_1.setExportFormattedValues(false);
ExportToCsv_1.setHierarchyLevelsInIndividualCells(true);
ExportToCsv_1.setScope(ExportScope.All);
ExportToCsv_1.exportReport();

```

5.38 API for Models with Accounts and Measures

To support models with accounts and measures, some new script APIs were introduced that deprecate previous script APIs.

5.38.1 Chart

In class `Chart` the following methods were deprecated and replaced:

Deprecated

```
addMeasure(measure: string | MeasureInfo, feed: Feed, position?: integer): void
```

Replaced by

```
addMember(feed: Feed, structureMember: string | MeasureInfo | MemberInfo,
position?: integer): void
```

Deprecated

```
getMeasures(feed: Feed): string[]
```

Replaced by

```
getMembers(feed: Feed): string[]
```

Deprecated

```
removeMeasure(measure: string | MeasureInfo, feed: Feed): void
```

Replaced by

```
removeMember(feed: Feed, member: string | MeasureInfo | MemberInfo): void
```

5.38.2 Data Explorer

In class `DataExplorer` the following method was deprecated and replaced:

Deprecated

```
setAdditionalMeasures(additionalMeasures: string[] | MeasureInfo[]): void
```

Replaced by

```
setAdditionalStructureDimensionMembers(structureDimension: string | DimensionInfo,
structureDimensionMembers: string[] | MemberInfo[] | MeasureInfo[]): void
```

5.38.3 Smart Discovery

Class `SmartDiscoveryMeasureSettings` was deprecated and its functionality replaced by class `SmartDiscoveryStructureSettings`, which provides the following script API:

```
create(dataSource: DataSource, target: string | MemberInfo | MeasureInfo,  
entityDimensions: string[] | DimensionInfo[], targetContext?: string | MemberInfo |  
MeasureInfo) : SmartDiscoveryStructureSettings  
copyDimensionFilterFrom(sourceDataSource: DataSource, dimension?: string |  
DimensionInfo): void  
setPageFilter(dimension: string | DimensionInfo, member: string | MemberInfo |  
string[] | MemberInfo[]): void  
setIncludedDataColumns(structureMembers: string[] | MemberInfo[] | MeasureInfo[]):  
void  
setIncludedDimensionColumns(includedDimensions: string[] | DimensionInfo[]): void  
setSettingsPanelVisible(settingsPanelVisible: boolean): void  
setVersion(version: string | MemberInfo): void
```

In class `SmartDiscoveryDimensionSettings` the following methods were deprecated and replaced:

Deprecated

```
setIncludedMeasures(includedMeasures: string[] | MeasureInfo[]): void
```

Replaced by

```
setIncludedDataColumns(structureMembers: string[] | MemberInfo[] | MeasureInfo[]):  
void
```

Deprecated

```
setIncludedDimensions(includedDimensions: string[] | DimensionInfo[]): void
```

Replaced by

```
setIncludedDimensionColumns(includedDimensions: string[] | DimensionInfo[]): void
```

5.38.4 Smart Grouping

In class `SmartGouping` the following method was deprecated and replaced:

Deprecated

```
setTooltipMeasureIncluded(included: boolean): void
```

Replaced by

```
setTooltipFeedsIncluded(included: boolean): void
```

5.39 Open in New Story API

For each data-bound widget at runtime, such as a Table or Chart, end users can create a new story from the widget and start exploration based on it afterwards.

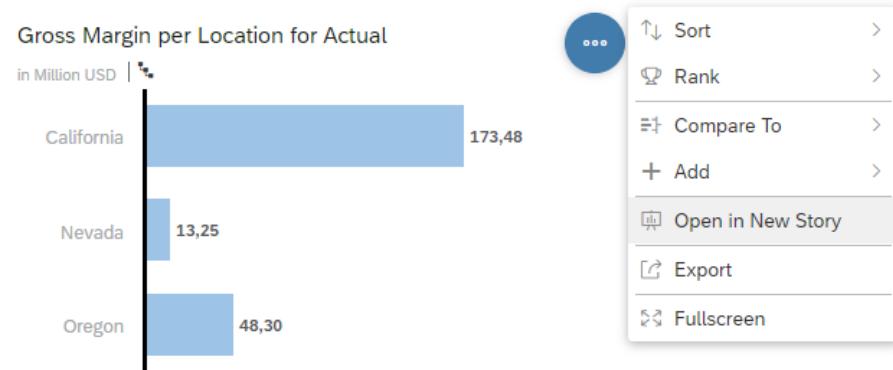


Figure 57: Create a Story from a Widget

The new story is created in a new browser page. The settings and data state (that is, filters and so on) will be carried over as well.

5.39.1 Script API

Besides selecting *Open in New Story* from the context menu of certain widgets, analytic application developers can use a script API to open a new story from an existing widget at runtime. The new story contains a copy of the widget in a new browser page. The supported widgets are the following:

- Chart
- Geo Map
- R Visualization
- Table

Example:

In the following example, new stories are created from existing widgets:

```
Chart_1.createStoryFromWidget();
Table_1.createStoryFromWidget();
RVisualization_1.createStoryFromWidget();
GeoMap_1.createStoryFromWidget();
```

5.40 Navigation API

The Navigation script API lets users navigate from an opened analytic application to another page of a story.

Basically, the script API can be used in two ways: open the analytic application or a page of a story directly or open an URL.

Navigate to Analytic Application or Story

The script API takes the uuid of an analytic application or a page in a story and opens the expected application or page in a new tab if parameter “newTab” is set to true.

```
NavigationUtils.openStory("story_uuid", "page_uuid",
[UrlParameter.create("p_script_var_1", "123"),
UrlParameter.create("p_script_var_2", "Value with Spaces")]);
NavigationUtils.openApplication("application_uuid", true);
```

Incidentally, when using `openStory()`, then the `page_uuid` is simply the page number as a string. For example, the `page_uuid` of the first page of a multi-page story is `"1"`.

Open URL

The user can also choose to open an URL, which is a story or analytic application URL, or even a general external URL. The URL can be opened in a new tab or in a browser page that's already open.

```
var storyURL = createStoryUrl("story_uuid", "page_uuid",
UrlParameter.create("p_script_var_1", "123"));
var appURL = createApplicationUrl("application_uuid");
openUrl(storyURL, true);
openUrl(appURL, true);
```

Open Data Analyzer

The user can also choose to open a data analyzer to analyze data of a data source. The user can pass the name of the connection, the name of the data source, and URL parameters, if necessary. The data analyzer opens in a new tab or in a browser page that's already open.

```
NavigationUtils.openDataAnalyzer("myconnection", "mydataSourceName",
UrlParameter.create("P_script_var_1", "123"), true);
```

5.41 Web Page

Using the Web Page widget, you can embed a web page into your analytic application to combine your analytics content with additional hosted live content.

5.41.1 Using Sandbox Restrictions

The Web Page widget is based on the `<iframe>` HTML element. You, as an analytic application developer, can apply additional sandbox restrictions to allow a greater degree of freedom for the embedded web page by selecting `allow-downloads` and `allow-popups` in the *Builder* panel.

Check option `allow-popups` when the Web Page widget embeds another analytic application or story that has visualizations connecting to SAP HANA or SAP BW Live data.

Note: If the live data connection type is *Tunnel*, you don't need to check option `allow-popups`.

The screenshot shows the SAP Analytics Cloud Builder interface. At the top, there's a blue header bar with the 'Builder' logo and a search icon. Below the header, a message states: 'In order to embed web pages into SAP Analytics Cloud, the website must grant access to all external sites or specifically trust the SAP Analytics Cloud domain.' A link 'What kind of web pages can I embed?' is provided. The main area is titled 'Web Page'. It includes fields for 'Web Page Title' (containing 'Web Page') and 'Web Page Address' (containing 'https://www.example.com'). Below these are 'Additional Sandbox Restrictions' settings, which include a dropdown menu with options: 'allow-popups' (selected), 'allow-downloads' (unchecked), and 'allow-scripts' (unchecked). A small info icon is also present.

In your analytic application you can use as many Web Page widgets as you like to embed other analytic applications or stories. However, check for potential issues such as performance and security considerations in the specification of the <iframe> HTML element.

Known restriction

Embedding an analytic application via a Web Page widget into stories or the digital boardroom isn't supported.

5.42 Dialogs to Share Applications and Bookmarks

You, as an end user, can open dialogs to share analytic applications and bookmarks.

You, as an analytic application developer, can open these dialogs at runtime with a script API.

5.42.1 Customize Link to Shared Application or Bookmark

You, as an end user, can edit the default link to the shared analytic application or bookmark in the dialog.

You, as an end user, can also add and edit URL parameters of the default link. The resulting updated default link is displayed in the *Default Link* text field.

The screenshots show the 'Share Bookmark' dialog box. In the top screenshot, the bookmark name is 'Bookmark_New Analytic Application' and the default link is a long URL. A 'Customize Link' button is visible. A modal window titled 'Add Users or Teams' is open, containing a text input field and a 'Share' button. In the bottom screenshot, the bookmark name is 'bk1', the custom link parameter is 'mode=embed', and there is a 'Done' button at the bottom.

Note: Some URL parameters can't be customized, such as the following:

- `appId`
- `view_id`

Note: The bookmark ID can't be changed when sharing a bookmark.

Note: The overall length of the default link is limited to 5000 characters. The length of the URL parameters contained in the default link is limited to 4000 characters.

Example:

The following link represents a default link to an analytic application:

<https://www.example.com/sap/.../7C51AB04E6C74174D9ACDEF133B09D40?mode=present>

Example:

Enter in the field Set URL Parameters `mode=embed`

Enter in the field the custom link `link/app1`

then this represents the following link:

<https://www.example.com/sap/.../7C51AB04E6C74174D9ACDEF133B09D40?mode=embed>

Example:

If you add the URL parameter `?p_gvar1=123` to the custom link at runtime as follows:

https://www.example.com/link/app1?p_gvar1=123

then this represents the following link:

https://www.example.com/sap/.../7C51AB04E6C74174D9ACDEF133B09D40?mode=embed&p_gvar1=123

Example:

If you add the URL parameters `?p_gvar1=123&mode=view` to the custom link at runtime as follows:

https://www.example.com/link/app1?p_gvar1=123&mode=view

then this represents the following link:

https://www.sampleexample.com/sap/.../7C51AB04E6C74174D9ACDEF133B09D40?mode=view&p_gvar1=123

5.42.2 Script API

5.42.2.1 Open Dialog to Share Analytic Application

With the following API you can open the dialog to share analytic applications at runtime:

`Application.openShareApplicationDialog(): void`

Note: You can share only the currently opened analytic applications.

5.42.2.2 Open Dialog to Share Personal Bookmark

With the following API you can open the dialog to share a personal bookmark at runtime:

`BookmarkSet_1.openShareBookmarkDialog(bookmarkId : string): void`

The bookmark name is displayed in the dialog to let you, as an end user, know which bookmark will be shared.

Note: You can share only personal bookmarks of the currently open analytic application.

5.43 Generic Widget Access API

You, as an analytic application developer, can use the following script API to retrieve all widgets of a specific type, for example, Table, Chart, or Button widgets. Optionally, you can specify a search pattern which retrieves only those widgets of a specific type that contain the text of the search pattern in their names:

`Application.getWidgets({ type: WidgetType, searchPattern?: string });`

Note: The search pattern is case-sensitive. A search pattern of "SalesChart" returns different widgets from the search pattern "saleschart".

Example:

With the following sample code snippet, you can retrieve an array of all Chart widgets of your application:

```
var charts = Application.getWidgets({ type: WidgetType.Chart });
```

Example:

With the following sample code snippet, you can retrieve an array of all Chart widgets of your application that contain the text "Sales" in their name:

```
var charts = Application.getWidgets({ type: WidgetType.Chart, searchPattern: "Sales" });
```

5.44 Input Control API

Get the Active Dimension Members Selected from Input Controls

As an application designer you can use the `getActiveSelectedMember` API to get the active dimension members selected from input controls.

```
getActiveSelectedMembers(options?: integer): MemberInfo[]
```

Note:

- API isn't supported for excluded members selected from input controls.
- For performance consideration, we recommend using it for input controls with low cardinality dimensions or turning on cascading effect to narrow down dimension members in input controls.
- `options` defines how many members are returned at most. By default it's 100 if not specified.

Example: use Input Control's values for Data Action Trigger

```
Var selectedMembers =
InputControl_1.getInputControlDataSource().getActiveSelectedMembers();

var arr = ArrayUtils.create(Type.string);
for (var i = 0; i < selectedMembers.length; i++){
    arr.push(selectedMembers[i].description);
}
DataAction_1.setParameterValue("CopyCountry", arr );
DataAction_1.execute();
Table_1.getDataSource().refreshData();
```

6 Typical Patterns and Best Practices

6.1 Switching Between Chart and Table

In this example, we will explore how to switch between a Chart and a Table using a toggle feature in an analytic application.

To achieve this, we will add an icon that represents a Chart and another that represents a Table. Then, we will write scripts for each of the images/icon we added to make it so that when we click on the Chart icon, the chart will appear, and the Table will be invisible, and vice versa.

Our default setting, shown when the application is first run, will be to make the Table visible (and the Chart invisible).

The result will look like this when we first run the application:

	Gross Margin Plan	Gross Margin	Gross Margin abs Dev	Gross Margin % Dev
> California	170,062	173,482	3,420	2.01 %
> Nevada	16,255	13,254	-3,001	-18.46 %
> Oregon	39,930	48,302	8,372	20.97 %

Figure 58: Example Application Switch Chart Table

And if we click on the image, we will get the Chart and the image will change its look to a and if we select it we come back to the view of the previous screenshot:

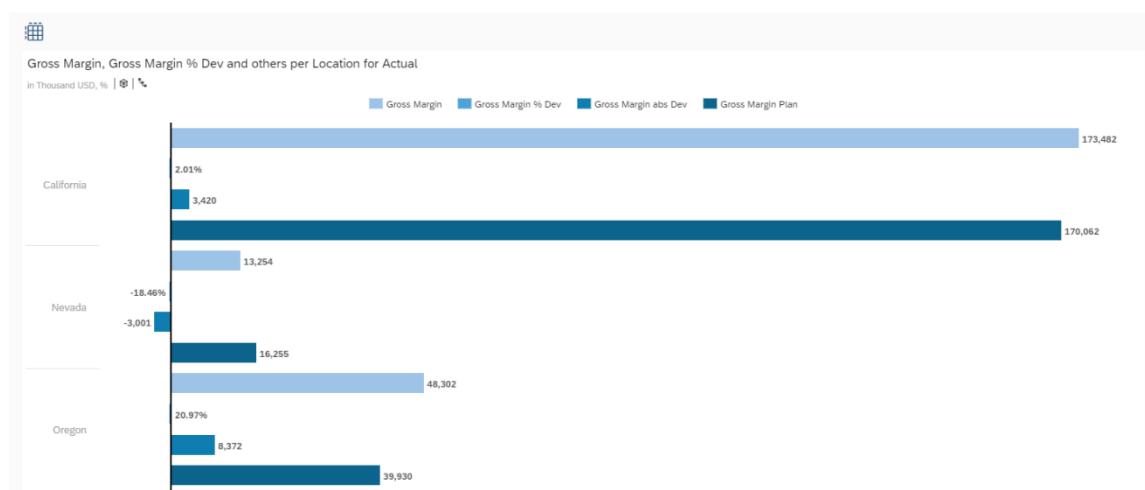
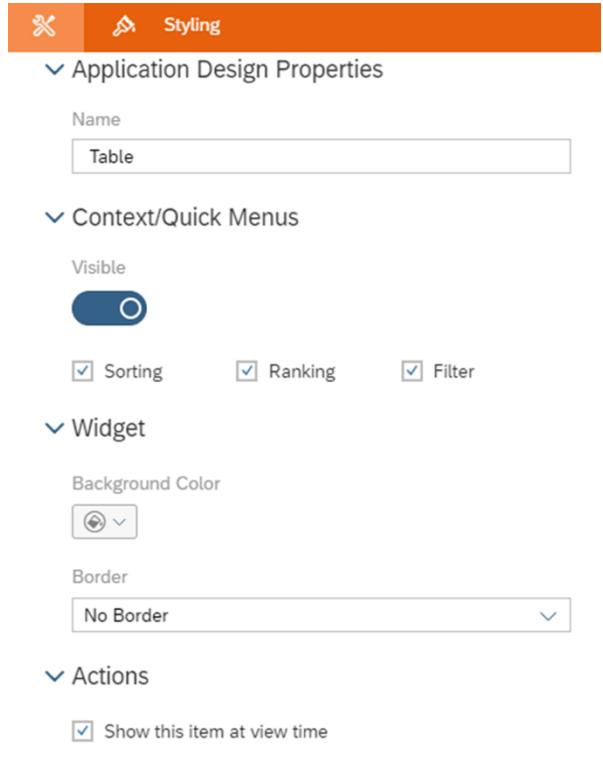
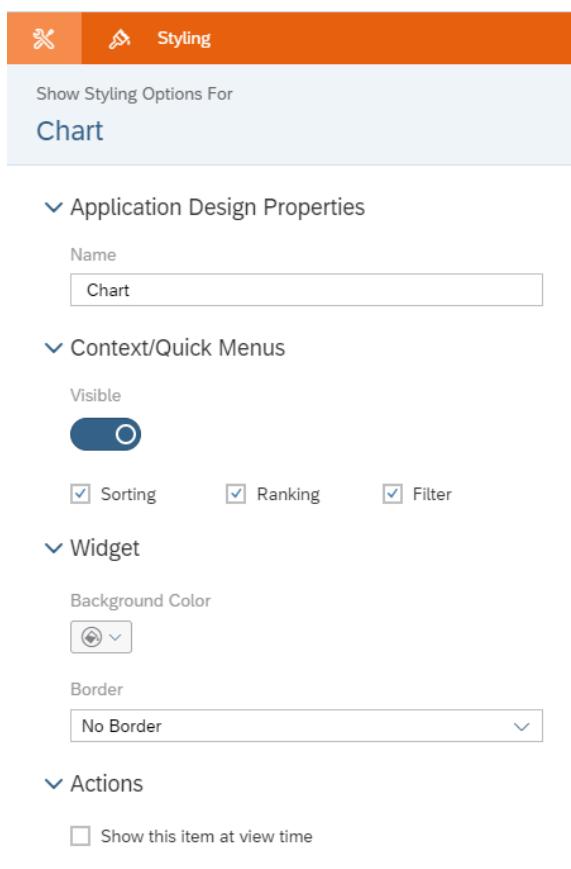
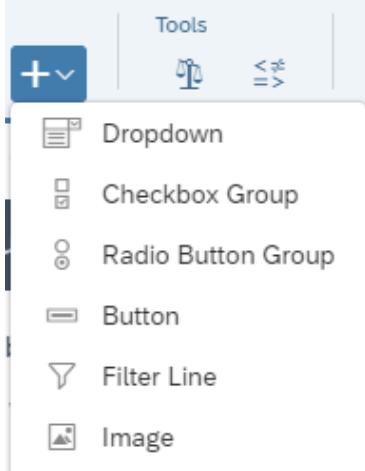
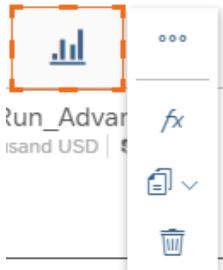
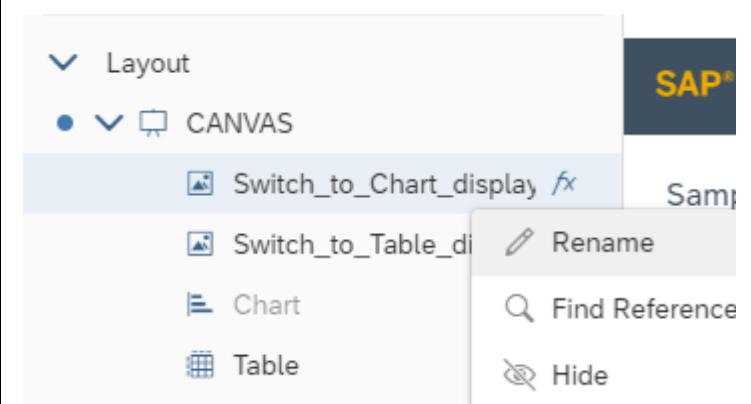
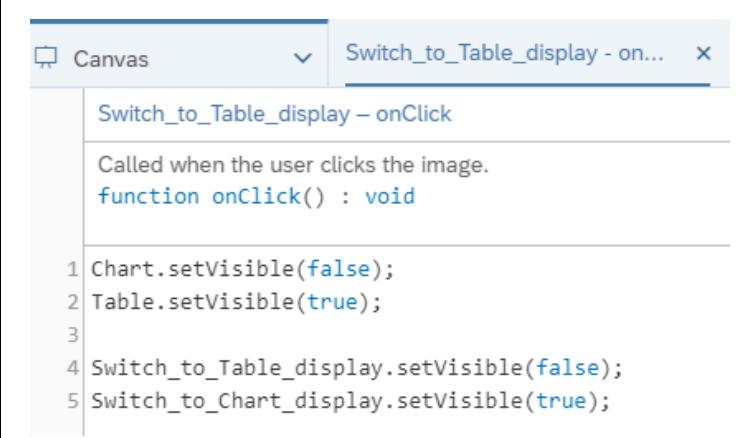


Figure 59: Switch Chart Table

Prerequisites for this use case is having already added a Table and a Chart to your Canvas. Select, for example, the model *BestRun_Advanced* as data source.

<p>Select the Table in your Canvas and click on Designer. Go to the <i>Styling</i> panel and under Actions, select "Show this item at view time".</p> <p>Afterwards, change the name of the widget to Table.</p>	
<p>Select the Chart afterwards and make sure that the action "Show this item at view time" is deselected.</p> <p>Afterwards, we will do the same as in the Table and change the name of this widget to "Chart"</p>	

<p>Choose the images you want the user to click on to change from Table to Chart and back.</p> <p>Here,  and  were used. You can insert them on top of each other so that when one is clicked on, the other one will appear in the same place.</p> <p>To insert an image, go to the <i>Insert</i> panel and under the “+” icon, select Image.</p>	
<p>To enable the switch between table and chart, we will edit the name and then the scripts of both images.</p> <p>First, we will edit the Chart Image's script.</p> <p>Select the image of the Chart you added and click on the .</p>	

<p>This will open the <code>onClick</code> event script of the image. Here, we will write a script that makes it possible to switch from the Table to the Chart. We have set the name of the  icon to <code>Switch_to_Table_display</code> and the  icon to <code>Switch_to_Chart_display</code>. This can be done through the Canvas in the Layout or Styling panel.</p> <p>This script makes the Chart visible and the Table invisible as well as set the visibility of the Table icon to true and the visibility of the Chart icon to false. This way when the Chart is visible, the icon of the Table will also be visible to indicate our ability to now switch back to the Table.</p>	 <div data-bbox="652 653 1403 1282"> <p><code>Canvas</code> <code>Switch_to_Chart_display - onClick</code></p> <p>Called when the user clicks the image.</p> <pre>function onClick() : void 1 Chart.setVisible(true); 2 Table.setVisible(false); 3 4 Switch_to_Table_display.setVisible(true); 5 Switch_to_Chart_display.setVisible(false); Chart.setVisible(true); Table.setVisible(false); Switch_to_Table_display.setVisible(true); Switch_to_Chart_display.setVisible(false);</pre> </div>
<p>We will now do the same for the icon of the Table. Select the image of the Table you chose and click on the  button.</p> <p>Here, we will set the Chart as well as the <code>Switch_to_Table_display</code> to false and the Table as well as the <code>Switch_to_Chart_display</code> to true.</p>	 <div data-bbox="652 1731 1403 1897"> <p><code>Canvas</code> <code>Switch_to_Table_display - onClick</code></p> <p>Called when the user clicks the image.</p> <pre>function onClick() : void 1 Chart.setVisible(false); 2 Table.setVisible(true); 3 4 Switch_to_Chart_display.setVisible(false); 5 Switch_to_Table_display.setVisible(true); Chart.setVisible(false); Table.setVisible(true); Switch_to_Chart_display.setVisible(false); Switch_to_Table_display.setVisible(true);</pre> </div>

Save the application and click on Run Analytic Application in the upper right side of the page and the result should look something like this:

The Table is displayed as we have set it to the default.

Now click on the Chart icon above it and the Table will change into a Chart.

If we want to look at the Table again, we only have to click on the icon of the Table and we would have the previous icon view again.

The screenshot shows two views of the same data. On the left is a table titled 'BestRun_Advanced' with data for California, Nevada, and Oregon across four columns: Gross Margin Plan, Gross Margin, Gross Margin abs Dev, and Gross Margin % Dev. On the right is a bar chart titled 'Gross Margin, Gross Margin % Dev and others per Location for Actual' showing the same data as the table, with bars for each location and their respective values labeled on top.

	Gross Margin Plan	Gross Margin	Gross Margin abs Dev	Gross Margin % Dev
California	170,062	173,482	3,420	2.01 %
Nevada	16,255	13,254	-3,001	-18.46 %
Oregon	39,930	46,302	6,372	20.97 %

6.2 Selecting Measures via Dropdown or Radio Button to Filter Table and Chart to Display (Single Selection)

In this example, we will explore how to filter a Table or a Chart using a single measure selected from a Dropdown widget or a Radio Button.

In the Dropdown widget, we will load all the measures from our data set and set the default filtering measure of the table to "Gross Margin Plan".

When another measure is selected, the filter is applied to the Table as well as the Chart (You can

go from the Table to the Chart and vice versa using the and the icons, respectively.)

The result will look like this when we run the application:

SAP Analytics Cloud

Sample - Selecting Measures via dropdown or radiobutton to filter table and chart to display (single selection)

Short Description

Selected Measure

BestRun_Advanced in Thousand USD |

	Gross Margin Plan
> California	170,062
> Nevada	16,255
> Oregon	39,930

Figure 60: Example Application Dropdown

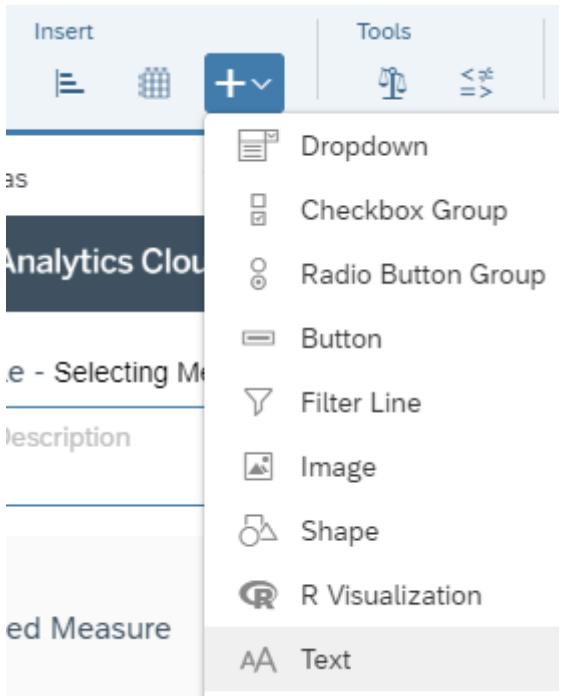
And if we click on the Dropdown, we will get all the measures with which we can filter the results of the Table or the Chart:

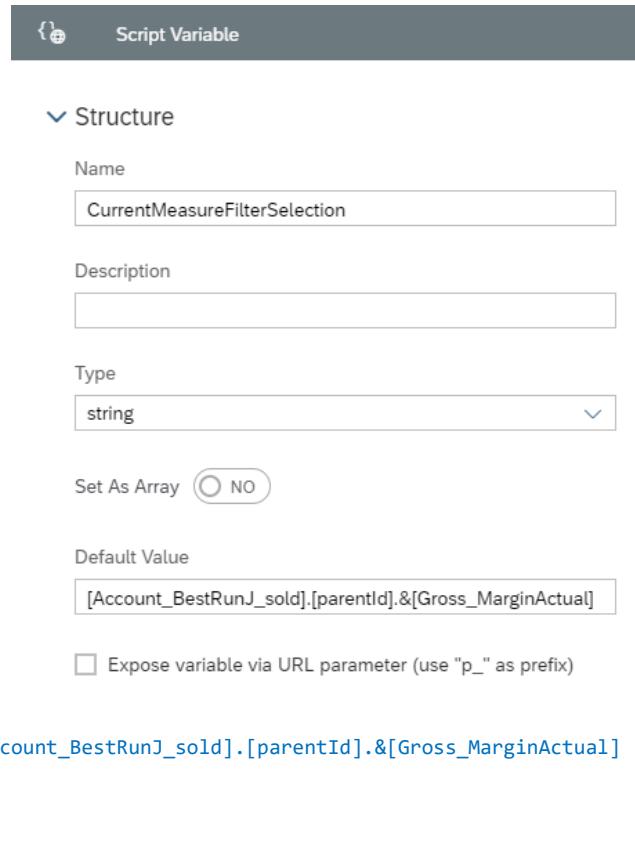
	Gross Margin Plan
California	170,062
Nevada	16,255
Oregon	39,930

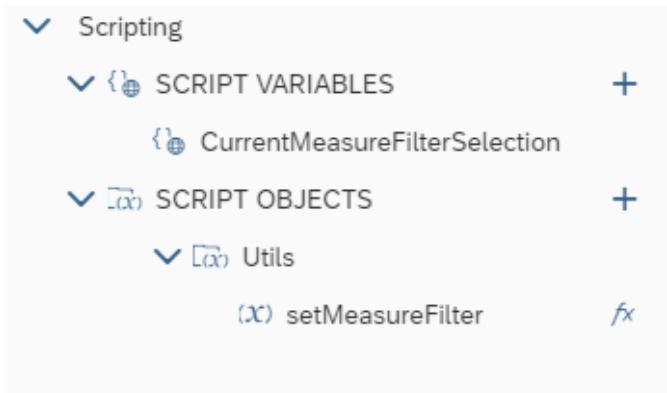
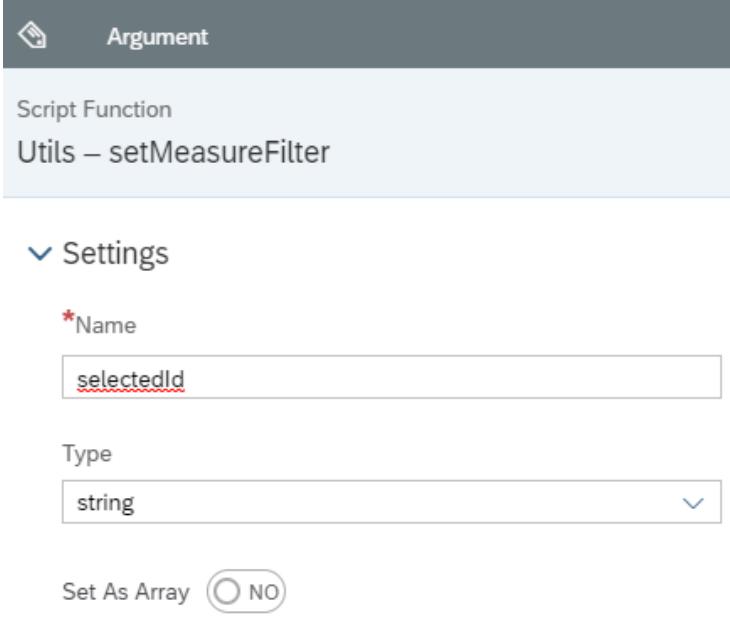
Figure 61: Dropdown Selection

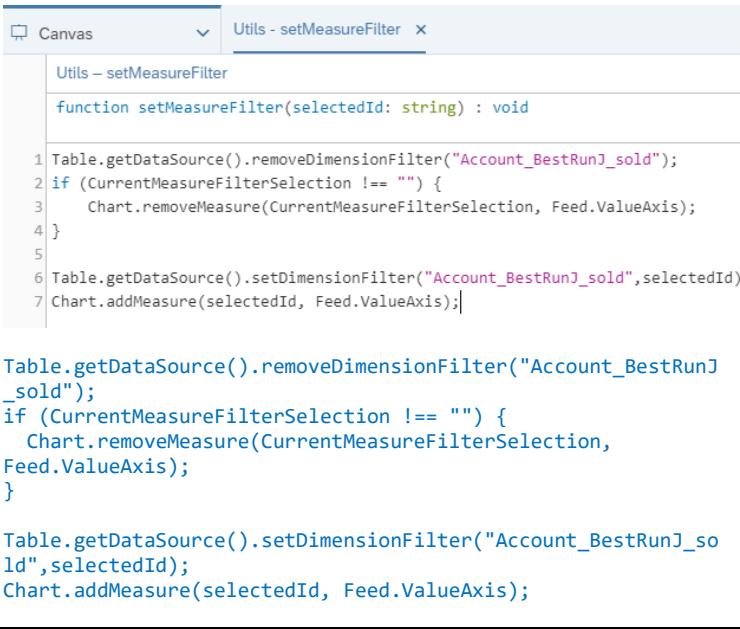
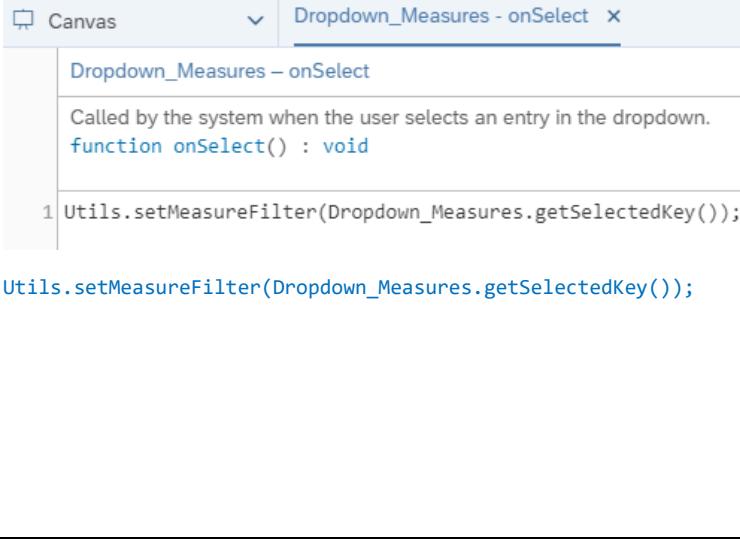
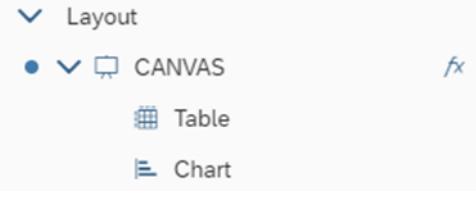
Prerequisites for this use case is having already added a table and a chart to your Canvas. To have all the functionalities in this use case, first go through the [Switching Between Chart and Table](#) exercise.

<p>To add a Dropdown widget, go to the <i>Insert</i> panel and click on the + sign and then choose <i>Dropdown</i>.</p> <p>We will name the dropdown <i>Dropdown_Measures</i>.</p> <p>To rename the objects, hover over them one by one in the Layout and when the icon appears click on it and choose <i>Rename</i>.</p> <p>This use case assumes that you've a Table and a Chart already set in the Canvas. If you don't, go through the <u>Switching Between Chart and Table</u> exercise and keep the names as they're in that exercise so that it works here as well.</p>	
---	--

<p>We will also add a Dropdown label so that we can indicate to the user that they can select measures through the Dropdown table.</p> <p>To insert Text, click again on the + icon in the <i>Insert</i> panel and choose <i>Text</i>.</p>	
<p>Place the Text widget on the left side of the Dropdown widget and we can then choose what to write in the Text box we inserted. We can, for example, write “Selected Measure”.</p>	
<p>Now we want to be able to access the value that the user chooses from the Dropdown widget. That's why we will add a Script Variable that acts as a global variable that can be accessed from anywhere in our application.</p> <p>To add a script variable, click on the “+” next to SCRIPT VARIABLES that's under Scripting.</p>	

<p>A window for the newly added script variable should now open. In the Structure part, type in CurrentMeasureFilterSelection as the Name and set the Default Value to [Account_BestRunJ_sold].[parentId].&[Gross_MarginActual]. This will make Gross Margin appear as our Default Value in the Dropdown widget when we run our application.</p> <p>Click on Done button to close variable definition dialog.</p>	 <p>The screenshot shows the 'Script Variable' dialog box with the 'Structure' tab selected. The 'Name' field contains 'CurrentMeasureFilterSelection'. The 'Type' field is set to 'string'. The 'Default Value' field contains '[Account_BestRunJ_sold].[parentId].&[Gross_MarginActual]'. A note below the default value field indicates the equivalent expression: '[Account_BestRunJ_sold].[parentId].&[Gross_MarginActual]'.</p>
---	--

<p>To define what should happen when a filter is selected, we need to create a Script Object. In this object, we will write a function that sets the measure filter according to what the user has chosen from the Dropdown options.</p> <p>To create a Script Object, select the "+" icon next to SCRIPT OBJECTS under the Layout. Afterwards, rename both the folder that was created as well as the function.</p> <p>We will name the folder Utils and the function setMeasureFilter.</p> <p>To rename the objects, hover over them one by one and when the  icon appears click on it and choose <i>Rename</i>.</p>	
<p>Click on the function setMeasureFilter and when the Properties window opens, click on the "+" icon next to Arguments.</p> <p>We will add an argument with the name selectedId and the Type string. Click on Done.</p>	

<p>Now we can write the script for the function.</p> <p>Hover over the <code>setMeasureFunction</code> and click on the  icon that appears next to it.</p> <p>Here, we will define what happens to the Table and the Chart when a user selects a measure from the Dropdown list.</p> <p>We will remove any already set dimensions of the Table or measures of the Chart and then we will add the captured value as the new dimension and measure of the Table as well as the Chart.</p>	 <pre> Canvas < Utils - setMeasureFilter > Utils - setMeasureFilter function setMeasureFilter(selectedId: string) : void 1 Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold"); 2 if (CurrentMeasureFilterSelection !== "") { 3 Chart.removeMeasure(CurrentMeasureFilterSelection, Feed.ValueAxis); 4 } 5 6 Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold",selectedId) 7 Chart.addMeasure(selectedId, Feed.ValueAxis); Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold"); if (CurrentMeasureFilterSelection !== "") { Chart.removeMeasure(CurrentMeasureFilterSelection, Feed.ValueAxis); } Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold",selectedId); Chart.addMeasure(selectedId, Feed.ValueAxis); </pre>
<p>Now that we have defined how the Table and Chart would change, we will define how to pass the captured value to the <code>setMeasureFilter</code> function. This will be done through <code>onSelect</code> event script of the Dropdown widget.</p> <p>To open the <code>onSelect</code> event script, click on the  icon next to the Dropdown object in the layout.</p> <p>This script will get the selected value of the Dropdown list and pass it to the <code>setMeasureFilter</code> as a parameter.</p>	 <pre> Canvas < Dropdown_Measures - onSelect > Dropdown_Measures - onSelect Called by the system when the user selects an entry in the dropdown. function onSelect() : void 1 Utils.setMeasureFilter(Dropdown_Measures.getSelectedKey()); Utils.setMeasureFilter(Dropdown_Measures.getSelectedKey()); </pre>
<p>The last step is setting what happens when the application is first run. This is done through the <code>onInitialization</code> event script of the Canvas itself.</p> <p>To get to this script, hover over the CANVAS in the Layout and click on the  icon when it appears and select <code>onInitialization</code>.</p>	

In this use case, we want to make sure that on initialization, we load all the available measures of the Table into our Dropdown List. After doing that, we set the selected key to the first measure in that list and then we set our measure filter to that first measure in our list.



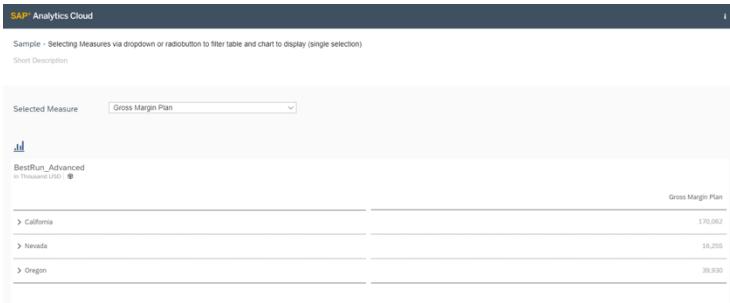
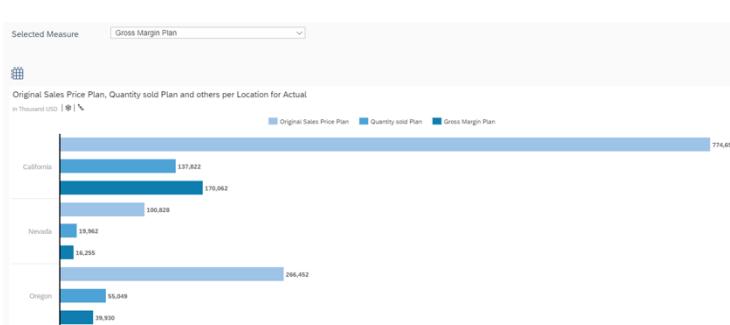
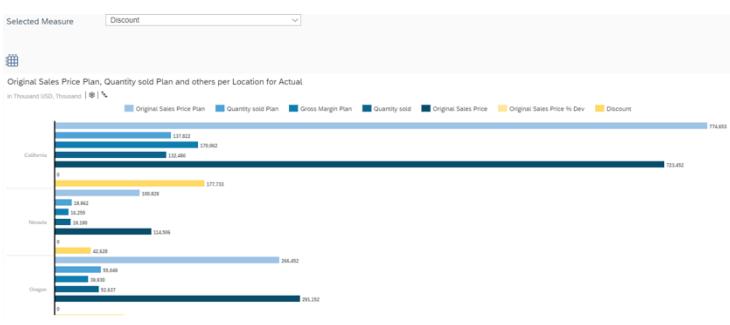
```
Application - onInitialization
Called when the Analytic Application has finished loading.
function onInitialization() : void

1 var measures = Table.getDataSource().getMeasures();
2
3 var selectedKey = "";
4
5 if (measures.length > 0) {
6     for (var i=0;i<measures.length; i++){
7         // Measure
8         Dropdown_Measures.addItem(measures[i].id,measures[i].description);
9         if (selectedKey === "" && i === 0) {
10             selectedKey = measures[i].id;
11             Dropdown_Measures.setSelectedKey(selectedKey);
12             console.log(["selectedKey ", selectedKey]);
13         }
14         console.log(["CurrentMeasure ", measures]);
15     }
16 }
17
18 Utils.setMeasureFilter(selectedKey);

var measures = Table.getDataSource().getMeasures();

var selectedKey = "";

if (measures.length > 0) {
    for (var i = 0; i < measures.length; i++) {
        // Measure
        Dropdown_Measures.addItem(measures[i].id,
measures[i].description);
        if (selectedKey === "" && i === 0) {
            selectedKey = measures[i].id;
            Dropdown_Measures.setSelectedKey(selectedKey);
            console.log(["selectedKey ", selectedKey]);
        }
        console.log(["CurrentMeasure ", measures]);
    }
}
Utils.setMeasureFilter(selectedKey);
```

<p>Now let's see how it looks like.</p> <p>Save the application and click on Run Analytic Application in the upper right side of the page and the result should look something like this:</p> <p>If you select a measure from the Dropdown list, the values in the Table as well as the Chart (accessed by clicking on the  icon – for more information, see exercise Switching Between Chart and Table) should change accordingly.</p>	<p>Application when it's first run:</p>  <p>Chart with "Gross Margin Plan" as the selected measure:</p>  <p>Chart with "Discount" as the selected measure:</p> 
--	---

6.3 Selecting Measures via Dropdown to Filter Table and Chart to Display (Multi-Selection)

In this example, we will explore how to filter a Table or a Chart using multiple measures selected from a Checkbox Group widget.

Unlike a Dropdown, the Checkbox Group allows using multiple measures as filters. In this use case, we will add a Checkbox Group widget where we will list all the measures in our data set. On top of that, there are the following three buttons:

- Set *selected* to filter the Table and Chart using the checked measures in the Checkbox
- Remove all to remove all the selected filters
- Set *all* to display all the available measures in our Table/Chart

The result will look like this when we run the application:

Sample - Selecting Measures via checkbox group to filter table and chart to display (multi selection)

Short Description

Measures

set selected | Remove all | set all

Gross Margin Plan
 Original Sales Price Plan
 Quantity sold Plan
 Gross Margin
 Original Sales Price
 Quantity sold
 Gross Margin abs Dev
 Gross Margin % Dev
 Original Sales Price abs Dev
 Original Sales Price % Dev
 Quantity sold abs Dev
 Quantity sold % Dev
 Discount
 Discount Plan
 Discount abs Dev
 Discount % Dev

BestRun_Advanced
In Thousand

	Gross Margin Plan	Original Sales Price Plan	Quantity sold Plan	Gross Margin	Original Sales Price	Quantity sold	Gross Margin abs Dev	Gross Margin % Dev
> California	\$170,062	\$774,653	\$137,822	\$173,482	\$723,452	\$132,480	\$3,420	2.01 %
> Nevada	\$16,255	\$100,828	\$19,962	\$13,254	\$114,506	\$19,190	-\$3,001	-18.46 %
> Oregon	\$39,930	\$266,452	\$55,049	\$48,302	\$291,152	\$52,637	\$8,372	20.97 %

Figure 62: Example Application Multi Selection

Prerequisites for this use case is having already added a table and a chart to your Canvas. To have all the functionalities in this use case, first go through the [Switching Between Chart and Table](#) exercise.

To add a Checkbox Group widget to your Canvas, go to the *Insert* panel and click on the “+” sign and then choose Checkbox Group. Place the newly added widget on the left side of your Table in the Canvas.

We will name the checkbox group `CheckboxGroup_Measures`.

To rename the objects, hover over them one by one and when the icon appears click on it and choose *Rename*.

Remove the initial values “Value 1” and “Value 2” from the Checkbox group Value

list. Select and click and then click *Apply*.

This use case assumes that you've a Table and a Chart already set in the Canvas. If you don't, go through the [Switching Between Chart and Table](#) exercise and keep the names as they're in that exercise so that it works here as well.



Dropdown



Checkbox Group



Radio Button Group



Button



Filter Line

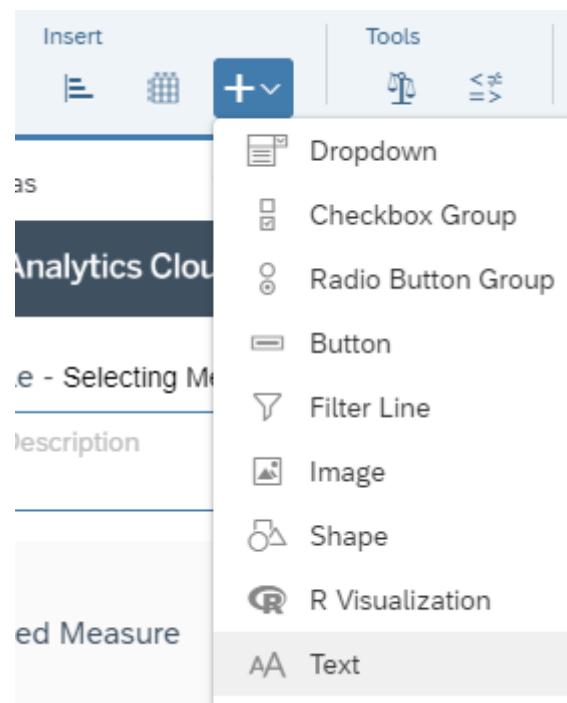
Builder

Checkbox Group Value

Value	Text (Optional)	Default
Value 1	<input checked="" type="checkbox"/>	
Value 2	<input type="checkbox"/>	

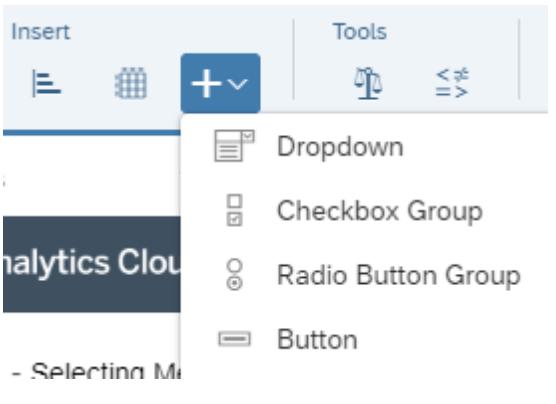
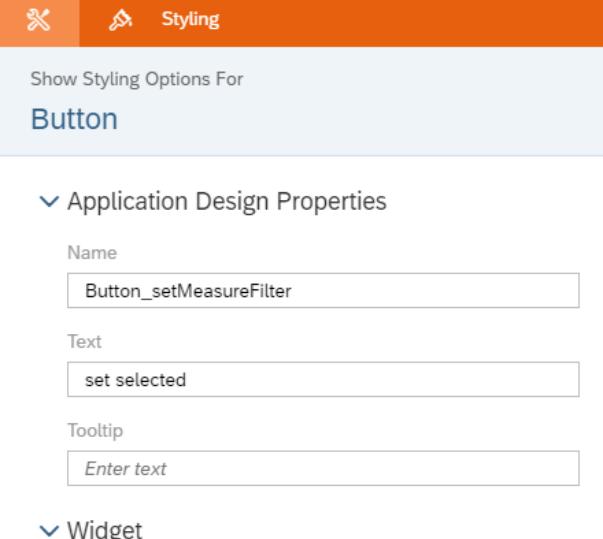
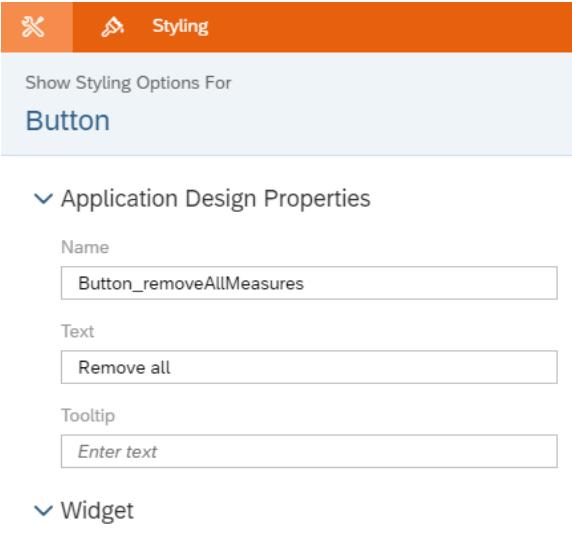
Apply

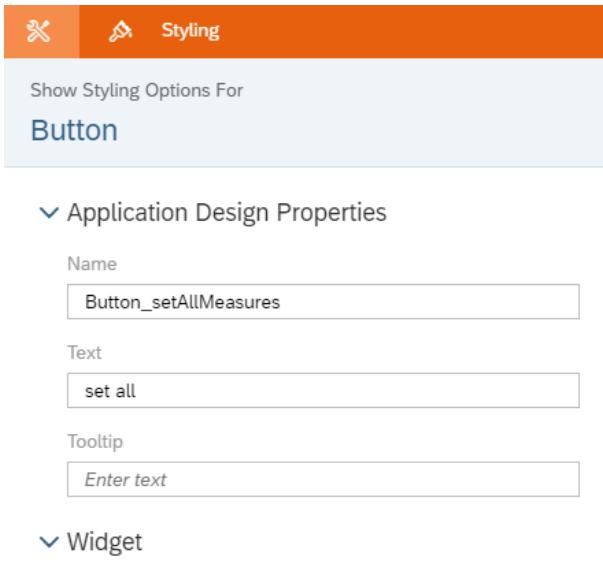
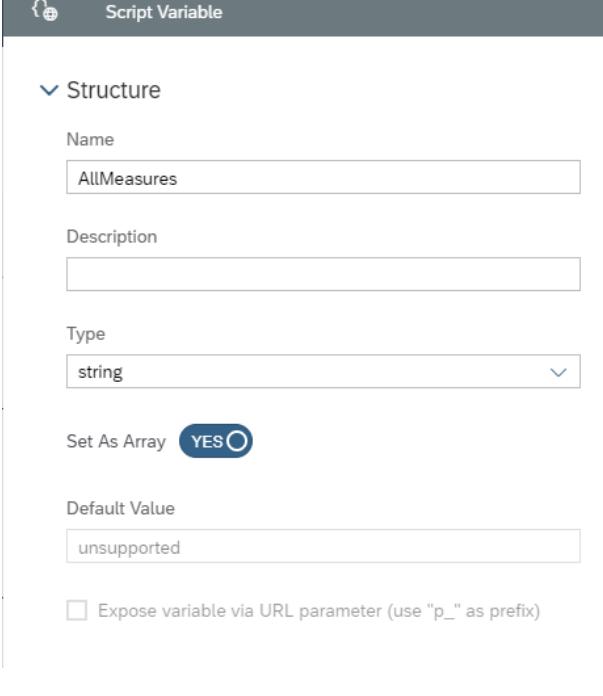
We will also add a label so that we can indicate to the user that the Checkbox Group is displaying the measures of our data set.
To insert Text, click again on the "+" icon in the *Insert* panel and choose *Text*.



Place the Text widget on the left side of the Dropdown widget and we can then choose what to write in the Text box we inserted. We can, for example, simple write "Measures".



<p>Now we want to be able to quickly use the Checkbox Group which is why we will add some buttons that will help us do that.</p> <p>The first button will be a “set selected” button; this will enable us to filter the data according to the selected checkboxes in our Checkbox Group.</p> <p>The second button will be a “Remove all” button; this will be a shortcut button that simplifies removing all the selected measures rather than deselecting them one by one.</p> <p>And the third, and final, button will be a “set all” button which when selected, selects all the measures in the Checkbox Group.</p> <p>To add the buttons, go to the “+” icon in the <i>Insert</i> panel and select Button and add three of them.</p>	
<p>After adding the three buttons, we will edit some of their properties.</p> <p>Select the first button and open the Designer (found on upper right part of the Page) and go to the <i>Styling</i> panel.</p> <p>There, change the name of the button to “Button_setMeasureFilter” and the Text to “set selected”.</p>	
<p>Select the second button and open the Designer again and go to the <i>Styling</i> panel.</p> <p>There, change the name of the button to “Button_removeAllMeasures” and the Text to “Remove all”.</p>	

<p>Select the third button and in the <i>Styling</i> panel, change the name of the button to "Button_setAllMeasures" and the Text to "set all".</p>	
<p>To be able to access the values that have been selected in the Checkbox Group, we need to create variables that can be accessed anywhere in the application. Which is why, we will create 2 Script Variables. The first one will be called "AllMeasures" and we will set it as an array. This variable will hold all the measures that could be selected in the Checkbox Group.</p> <p>To insert this variable, go to SCRIPT VARIABLES under SCRIPTING in the Layout which you can find on the left part of the Page. Click on the "+" icon next to the SCRIPTING VARIABLES which should open a new window where you can change the structure of your variable.</p> <p>There, type in "AllMeasures" in the Name box, select "string" as Type, and set the Set As Array button to "YES". Click on Done to close the properties' window.</p>	

<p>Add a second Scripting Variable the same way as in Step 8. This variable will hold the measures that the user has selected from the Checkbox Group.</p> <p>When the Structure window opens, type in "CurrentMeasureFilterSelection" in the Name box, set "string" as Type, and toggle the Set As Array button to "YES".</p>	<p>The screenshot shows the 'Script Variable' dialog box. At the top, there's a header bar with a save icon and the title 'Script Variable'. Below it is a section titled 'Structure' with a dropdown arrow. The 'Name' field contains 'CurrentMeasureFilterSelection'. The 'Type' dropdown menu is open, showing 'string' as the selected option. A 'Set As Array' button is shown in a blue circle with the word 'YES' in white. Below these are fields for 'Default Value' (containing 'unsupported') and a checkbox for 'Expose variable via URL parameter (use "p_" as prefix)' which is unchecked.</p>
<p>To define what should happen when a filter is selected, we need to create a Script Object.</p> <p>In this object, we will create a function that sets the measure filter according to what the user has chosen from the Checkbox Group.</p> <p>To create a Script Object, select the "+" icon next to SCRIPT OBJECTS under the Layout. Afterwards, rename both the folder that was created as well as the function.</p> <p>We will name the folder Utils and the function setMeasureFilter.</p> <p>To rename the objects, hover over them one by one and when the icon appears click on it and choose Rename.</p>	<p>The screenshot shows the 'Script Objects' tree view. It starts with a 'SCRIPT VARIABLES' node containing 'AllMeasures' and 'CurrentMeasureFilterSelector'. Below that is a 'SCRIPT OBJECTS' node containing a 'Utils' folder, which in turn contains a function named 'setMeasureFilter' with a delete icon next to it.</p>

Click on the function setMeasureFilter and when the Editing window opens, click on the “+” icon next to Arguments.

Here, we will add an argument with the name “selectedIds” and the Type string[] (array of strings).

(X) Script Function

Script Object
Utils

Properties

*Name: setMeasureFilter

Description:

Return Type: void

Set As Array: NO

Arguments +

Argument

Script Function
Utils – setMeasureFilter

Settings

*Name: selectedIds

Type: string

Set As Array: YES

To define what the `setMeasureFilter` function, that we added in Step 11, does, go to the function in the Layout, hover over its name, and click on the  icon next to it.

The script of this function does the following:
Firstly, it removes any dimensions from the Table or measures from the Chart that were added to filter them before.

Secondly, it looks to see which measures the user has chosen from the Checkbox Group and adds them as dimensions to the Table/measures for the Chart.

Lastly, it takes the selected measures of the user and saves them in the variable we created in Step 9 (`CurrentMeasureFilterSelection`).

```
Canvas Utils - setMeasureFilter x
Utils - setMeasureFilter
function setMeasureFilter(selectedIds: string[]): void

1 // remove Measures
2 Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold");
3 if (CurrentMeasureFilterSelection != "") {
4   for (var i=0;i<CurrentMeasureFilterSelection.length; i++){
5     Chart.removeMeasure(CurrentMeasureFilterSelection[i], Feed.ValueAxis);
6   }
7 }
8
9 // add Measures
10 Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold",selectedIds);
11 for (i=0;i<selectedIds.length; i++){
12   Chart.addMeasure(selectedIds[i], Feed.ValueAxis);
13 }
14
15 // save the current selection into global variable
16 CurrentMeasureFilterSelection = selectedIds;
```



```
// remove Measures
Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold");
if (CurrentMeasureFilterSelection != "") {
  for (var i = 0; i < CurrentMeasureFilterSelection.length; i++) {
    Chart.removeMeasure(CurrentMeasureFilterSelection[i], Feed.ValueAxis);
  }
}

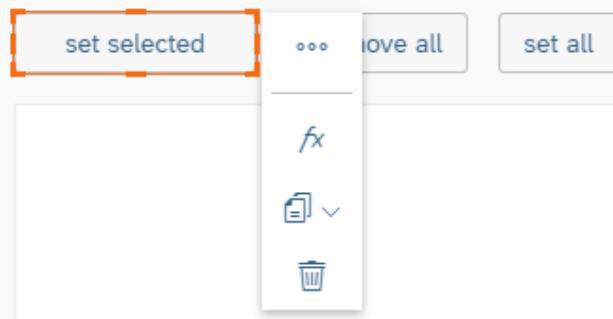
// add Measures
Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold",selectedIds);
for (i = 0; i < selectedIds.length; i++) {
  Chart.addMeasure(selectedIds[i], Feed.ValueAxis);
}
// save the current selection into global variable
CurrentMeasureFilterSelection = selectedIds;
```

Now, we need to define what happens when the buttons we created are clicked. The first button, "set selected" should filter the data according to the selected checkboxes in our Checkbox Group.

To edit the `onClick` event script of the button, you can either hover over it in the

Layout and click on the  icon or you can click on the button in the Canvas and similarly select .

Measures



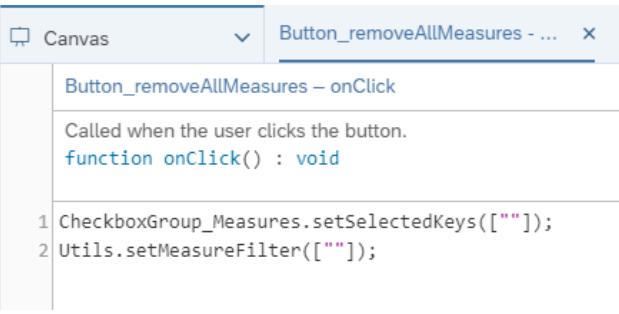
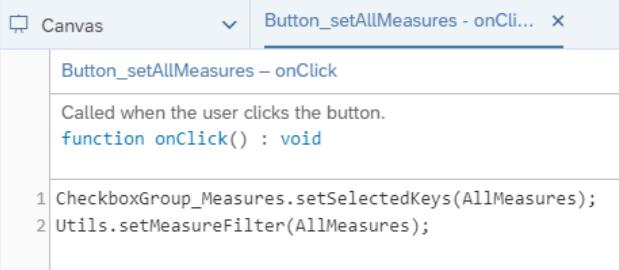
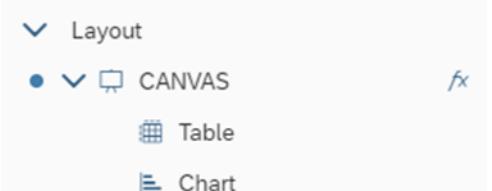
In the `onClick` event script, we will call the `Utils.setMeasureFilter` function and pass to it the selected measures of the Checkbox Group.

```
Canvas Button_setMeasureFilter - on...
Button_setMeasureFilter - onClick
Called when the user clicks the button.
function onClick(): void

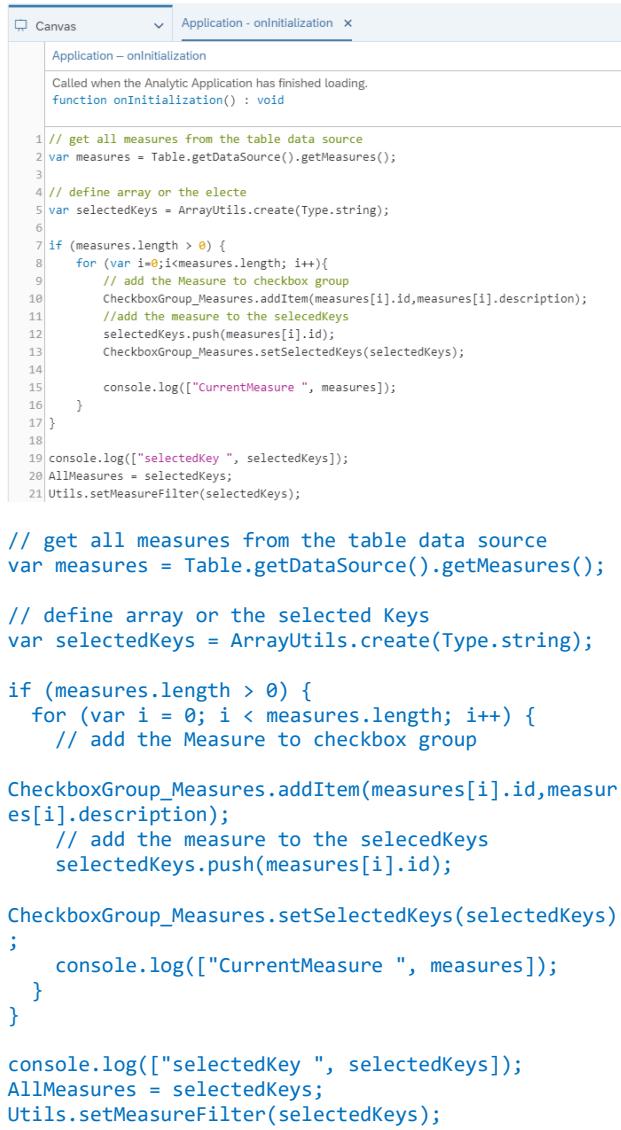
1 Utils.setMeasureFilter(CheckboxGroup_Measures.getSelectedKeys());
```



```
Utils.setMeasureFilter(CheckboxGroup_Measures.getSelectedKeys());
```

<p>The second button, “Remove all”, removes all the selected measures from the Checkbox Group.</p> <p>Open the script of the button like in step 13 and here, we will remove all the selected measures from the Checkbox Group itself and pass an empty array to the Utils.setMeasureFilter so that our Table and Chart as well as our global variable CurrentMeasureFilterSelection will be updated.</p>	 <pre>Button_removeAllMeasures - onClick Called when the user clicks the button. function onClick() : void 1 CheckboxGroup_Measures.setSelectedKeys([""]); 2 Utils.setMeasureFilter([""]); CheckboxGroup_Measures.setSelectedKeys([""]); Utils.setMeasureFilter([""]);</pre>
<p>The third button, “set all”, selects all the measures in the Checkbox Group.</p> <p>In the script of this button, we will set the selected keys of the Checkbox Group to the AllMeasures script variable we had defined before and we will pass the same variable to the Utils.setMeasureFilter function.</p>	 <pre>Button_setAllMeasures - onClick Called when the user clicks the button. function onClick() : void 1 CheckboxGroup_Measures.setSelectedKeys(AllMeasures); 2 Utils.setMeasureFilter(AllMeasures); CheckboxGroup_Measures.setSelectedKeys(AllMeasures); Utils.setMeasureFilter(AllMeasures);</pre>
<p>The last step is setting what happens when the application is first run. This is done through the <code>onInitialization</code> event script of the Canvas itself.</p> <p>To get to this script, hover over the CANVAS in the Layout and click on the  icon when it appears and select <code>onInitialization</code>.</p>	

In this use case, we want to make sure that on initialization, we get all the available measures of the Table's data source. Then, we define a selected keys array of type string and using a loop, we add the measures to our Checkbox Group and the selected keys array. We also call on the setSelectedKeys function of the Checkbox Group and set its selected keys to our array. Finally, we set the script variable AllMeasures and the measure filter to the selected keys.



```

1 // get all measures from the table data source
2 var measures = Table.getDataSource().getMeasures();
3
4 // define array or the electe
5 var selectedKeys = ArrayUtils.create(Type.string);
6
7 if (measures.length > 0) {
8     for (var i=0;i<measures.length; i++){
9         // add the Measure to checkbox group
10        CheckboxGroup_Measures.addItem(measures[i].id,measures[i].description);
11        //Add the measure to the selectedKeys
12        selectedKeys.push(measures[i].id);
13        CheckboxGroup_Measures.setSelectedKeys(selectedKeys);
14
15        console.log(["CurrentMeasure ", measures]);
16    }
17 }
18
19 console.log(["selectedKey ", selectedKeys]);
20 AllMeasures = selectedKeys;
21 Utils.setMeasureFilter(selectedKeys);

// get all measures from the table data source
var measures = Table.getDataSource().getMeasures();

// define array or the selected Keys
var selectedKeys = ArrayUtils.create(Type.string);

if (measures.length > 0) {
    for (var i = 0; i < measures.length; i++) {
        // add the Measure to checkbox group

CheckboxGroup_Measures.addItem(measures[i].id,measures[i].description);
        // add the measure to the selecedKeys
        selectedKeys.push(measures[i].id);

CheckboxGroup_Measures.setSelectedKeys(selectedKeys);
    ;
        console.log(["CurrentMeasure ", measures]);
    }
}

console.log(["selectedKey ", selectedKeys]);
AllMeasures = selectedKeys;
Utils.setMeasureFilter(selectedKeys);

```

Now let's see how it looks like.
Click on Run Analytic Application in the upper right side of the page and the result should look something like this:

If we click on the "Remove all" button, all measures are deselected and there is no Table (or Chart) -> accessed through the



icon) to look at.

If we click on "set all", all measures are selected again, and the Table (or Chart) looks like when we first ran the application.

Let us only select a few measures and see how the Table will change.

In the screenshot on the right, 4 measures are chosen (Gross Margin Plan, Quantity Sold, Original Sales Price abs Dev, and Discount).

After selecting the measures, click on "set selected" to update the Table/Chart with your chosen measures.

Application when it's first run:

	Gross Margin Plan	Original Sales Price Plan	Quantity sold Plan	Gross Margin	Original Sales Price	Quantity sold	Gross Margin abs Dev	Gross Margin % Dev
> California	\$170,000	\$171,400	\$123,450	\$123,450	\$122,480	\$12,400	2,01%	
> Nevada	\$16,200	\$16,300	\$16,000	\$16,300	\$16,400	\$10,000	-18,40%	
> Oregon	\$20,000	\$20,400	\$16,000	\$20,150	\$20,200	\$20,000	20,00%	

Table after clicking on "Remove all":

	Gross Margin Plan	Original Sales Price Plan	Quantity sold Plan	Gross Margin	Original Sales Price	Quantity sold	Gross Margin abs Dev	Gross Margin % Dev
> California								
> Nevada								
> Oregon								

Table after clicking on "set all":

	Gross Margin Plan	Original Sales Price Plan	Quantity sold Plan	Gross Margin	Original Sales Price	Quantity sold	Gross Margin abs Dev	Gross Margin % Dev
> California	\$170,000	\$171,400	\$123,450	\$123,450	\$122,480	\$12,400	2,01%	
> Nevada	\$16,200	\$16,300	\$16,000	\$16,300	\$16,400	\$10,000	-18,40%	
> Oregon	\$20,000	\$20,400	\$16,000	\$20,150	\$20,200	\$20,000	20,00%	

Table after selecting specific measures:

	Gross Margin Plan	Quantity sold	Original Sales Price abs Dev	Discount
> California	170,000	132,400	-12,200	17,733
> Nevada	16,200	16,000	1,600	42,626
> Oregon	20,000	16,000	4,000	82,770

6.4 Using Filter Line for Filtering Table, Chart, and R Visualization

In this example, we will explore how to filter a Table, a Chart or an R Visualization using a Filter Line widget.

Instead of loading all the dimensions in our data set into a Checkbox group or a Dropdown widget, in this use case, we will select specific dimensions to load into a Filter Line.

Unlike other data-bound widgets (such as Table or Chart), R Visualization can add multiple input data models. To support R Visualization in Filter Line, one Dropdown list is added to select the connected input data.

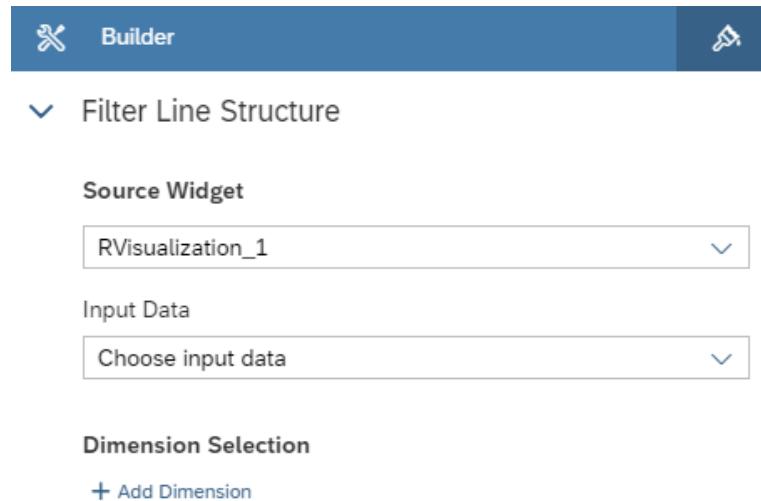


Figure 63: Choose Input Data for Filtering R Visualization

After the user selects an input data model of the R Visualization widget, the Filter Line can support R Visualization just like other widgets.

After loading the desired dimensions into our Filter Line, we will connect it to our Table/Chart/R Visualization so that the data is filtered using the selected filter.

To use the Filter Line after running the application, simply click on the Filter Line icon and select the dimension you want to use to filter your data.

The result will look like this when we run the application:

	Gross Margin Plan	Gross Margin	Gross Margin abs Dev	Gross Margin % Dev
> California	170,062	173,482	3,420	2.01 %
> Nevada	16,255	13,254	-3,001	-18.46 %
> Oregon	39,930	48,302	8,372	20.97 %

Figure 64: Example Application Filter Line

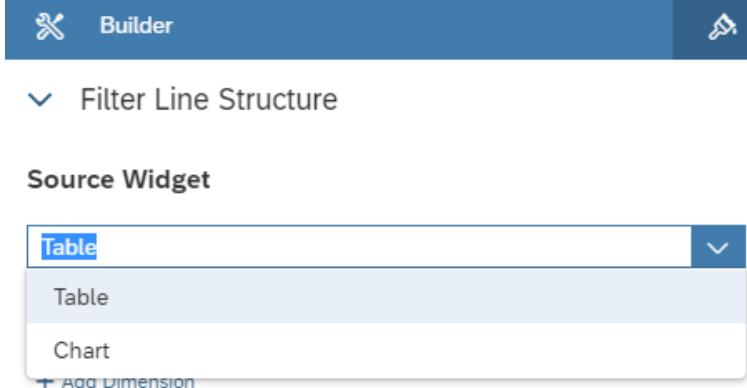
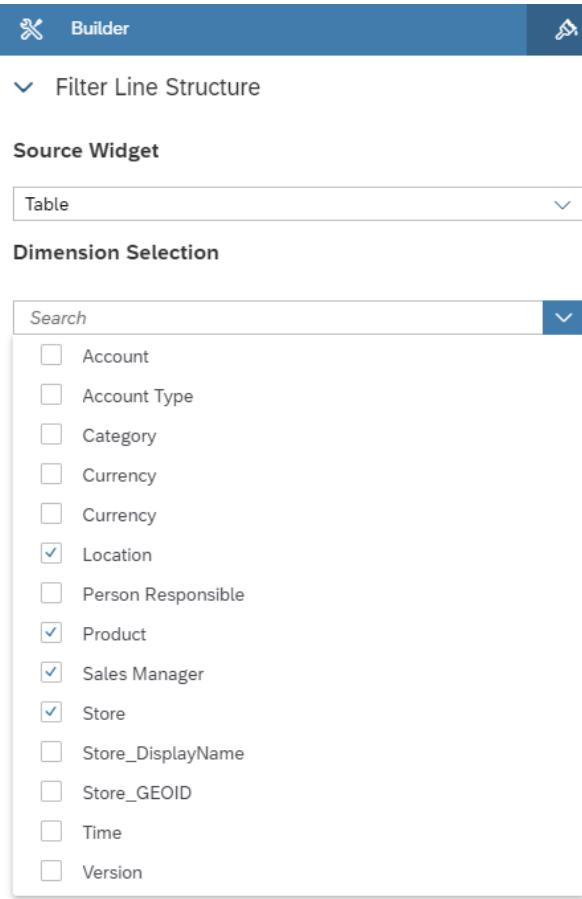
And this is how it will look like when we click on our Filter Line widget:

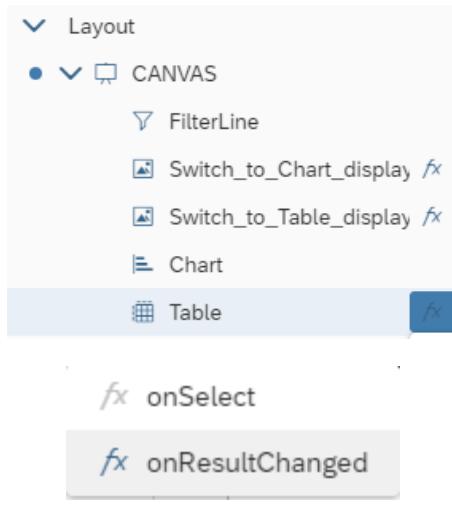
	Gross Margin abs Dev	Gross Margin % Dev	Gross Margin	Gross Margin Plan
> California	3,420	2.01 %	173,482	170,062
> Nevada	-3,001	-18.46 %	13,254	16,255
> Oregon	8,372	20.97 %	48,302	39,930

Figure 65: Select Filter Line

Prerequisites for this use case is having already added a Table and a Chart to your Canvas. To have all the functionalities in this use case, first go through the [Switching Between Chart and Table](#) exercise.

<p>To add a Filter Line widget to your Canvas, go to the <i>Insert</i> panel and click on the “+” sign and then choose <i>Filter Line</i>. Place the newly added widget above the Table.</p> <p>We will name the filter line <i>FilterLine</i>.</p> <p>To rename the objects, hover over them one by one and when the icon appears click on it and choose <i>Rename</i>.</p> <p>This use case assumes that you've a Table and a Chart already set in the Canvas. If you don't, go through the Switching Between Chart and Table exercise and keep the names as they're in that exercise so that it works here as well.</p>	<ul style="list-style-type: none"> Dropdown Checkbox Group Radio Button Group Button Filter Line
<p>After adding the Filter Line, we need to set its properties. We can do that by selecting the Filter Line we added to our Canvas and afterwards, clicking on the Designer button. You can find this button on the upper right side of the screen.</p> <p>There, navigate over to the <i>Builder</i> panel.</p>	<p> Styling</p>

<p>In the <i>Builder</i> panel, we will set the structure of the Filter Line. We will set the source widget as the Table. This is done by going to Source Widget and choosing "Table" from the Dropdown List.</p>	 <p>The screenshot shows the 'Builder' interface. Under 'Filter Line Structure', there is a 'Source Widget' section. A dropdown menu is open, showing 'Table' as the selected option, with other options like 'Chart' and a link to 'Add Dimension'.</p>
<p>Now we will add the filters we want: In this use case we want the user to be able to filter on 4 dimensions: Location, Product, Sales Manager, and Store. We can add these by going to the Dimension Selection part and clicking Add Dimension and selecting all 4 when the Checklist comes up.</p>	 <p>The screenshot shows the 'Builder' interface. Under 'Filter Line Structure', there is a 'Dimension Selection' section. A checklist is displayed with the following items, where 'Location', 'Product', 'Sales Manager', and 'Store' are checked:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Account <input type="checkbox"/> Account Type <input type="checkbox"/> Category <input type="checkbox"/> Currency <input type="checkbox"/> Currency <input checked="" type="checkbox"/> Location <input type="checkbox"/> Person Responsible <input checked="" type="checkbox"/> Product <input checked="" type="checkbox"/> Sales Manager <input checked="" type="checkbox"/> Store <input type="checkbox"/> Store_DisplayName <input type="checkbox"/> Store_GEOID <input type="checkbox"/> Time <input type="checkbox"/> Version

<p>In step 3 we needed to select a source widget for our Filter Line, and we chose the Table, however, in our application we give the user the option to toggle between Table and Chart using the  and  respectively (refer to the Switching Between Chart and Table Exercise).</p> <p>This means that we have to find a way to get the filter that's been applied to the Table so that we can apply that on our Chart too.</p> <p>To do that, click on the  next to the Table in the Layout and select <i>onResultChanged</i>.</p>	
	<p>In the onResultChanged event script, we will copy the dimension filters from the Table. We do the copying 4 times for each of the measures we had added in the Dimension Selection part (in step 4).</p> <pre data-bbox="666 875 1381 1370"> Table - onResultChanged Called when the result set displayed by the table changes. function onResultChanged() : void { console.log('OnResultChanged'); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Location_4nm2e04531"); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Product_3e315003an"); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Sales_Manager__5w3m5d06b5"); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Store_3z2g5g06m4.Store_GEOID"); console.log('OnResultChanged'); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Location_4nm2e04531"); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Product_3e315003an"); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Sales_Manager__5w3m5d06b5"); Chart.getDataSource().copyDimensionFilterFrom(Table.getDataSource(), "Store_3z2g5g06m4.Store_GEOID"); } </pre>

Now let's see how it looks like.

Click on Run Analytic Application in the upper right side of the page and the result should look something like this:

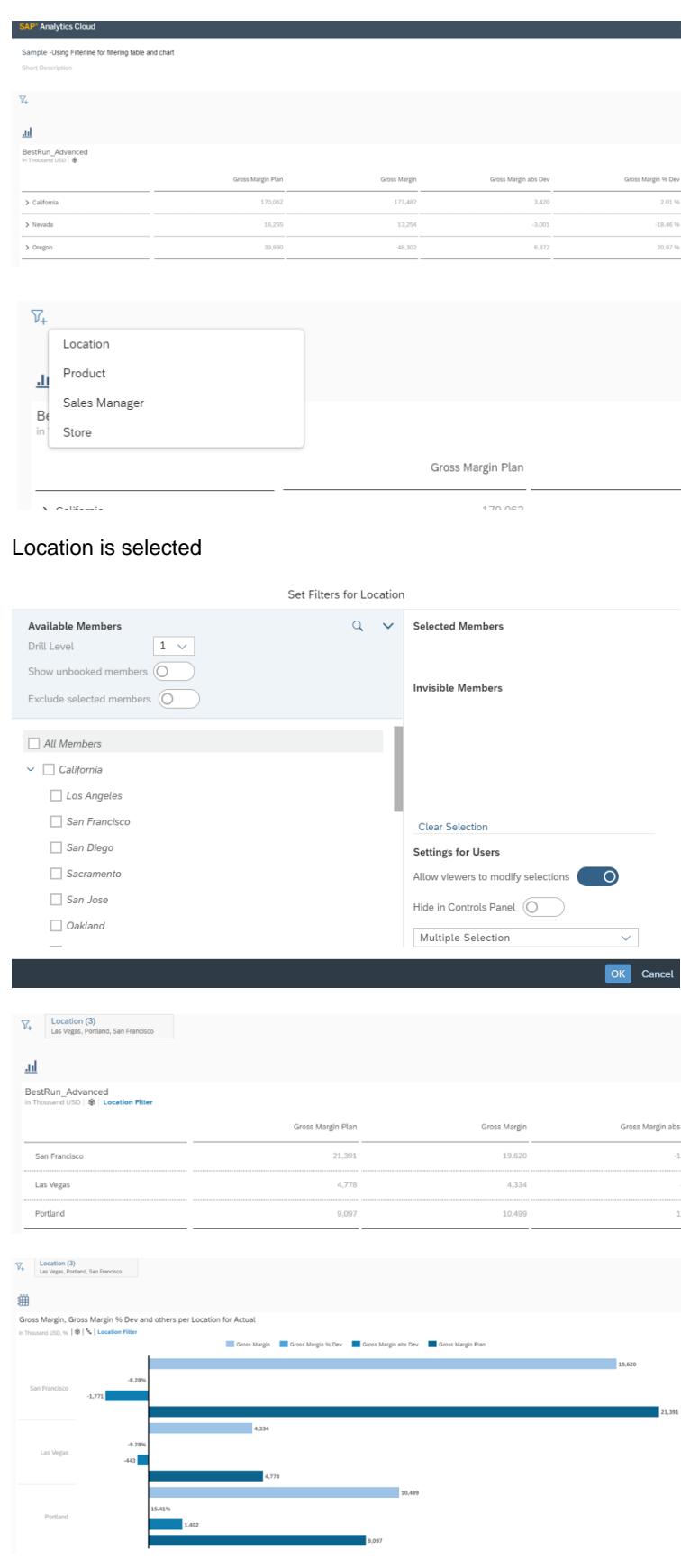
When you click on the Filter Line, the 4 measures we added pop up.

When one of the measures in the Filter Line is clicked, a pop-up window comes up and we get to choose which cities (locations), products, sales managers, and stores do we want to include in our Table or Chart.

If we were to choose *San Francisco, Las Vegas, and Portland* as our members, the table would update according to that filter.

And the Chart will be updated as

well (Click the  icon to get the view of the Chart).



The screenshot shows the SAP Analytics Cloud interface. At the top, there is a navigation bar with icons for Home, Overview, and Help. Below the navigation bar, the title "Sample - Using Filterline for filtering table and chart" is displayed. The main content area contains a table titled "BestRun_Advanced" (in Thousand USD). The table has four columns: Gross Margin Plan, Gross Margin, Gross Margin abs Dev, and Gross Margin % Dev. The data rows are California (170,062, 173,482, 3,400, 2.01%), Nevada (16,255, 13,254, -3,001, -18.66%), and Oregon (39,930, 48,302, 8,372, 20.97%).

Below the table, a "Location Filter" dialog is open. It lists categories: Location, Product, Sales Manager, and Be in Store. Under Location, the options are Location, Product, Sales Manager, and Be in Store. A sub-dialog titled "Gross Margin Plan" is also visible.

The text "Location is selected" is displayed above the filter dialog. The filter dialog itself is titled "Set Filters for Location". It includes sections for "Available Members" (with Drill Level set to 1) and "Selected Members" (which is currently empty). The "Available Members" section lists various locations: All Members, California, Los Angeles, San Francisco, San Diego, Sacramento, San Jose, and Oakland. The "Selected Members" section is labeled "Invisible Members".

At the bottom of the filter dialog, there are buttons for "OK" and "Cancel".

At the bottom of the main screen, there is a chart titled "Gross Margin, Gross Margin % Dev and others per Location for Actual" (in Thousand USD). The chart shows four bars for each location: Gross Margin (light blue), Gross Margin % Dev (dark blue), Gross Margin abs Dev (medium blue), and Gross Margin Plan (dark blue). The values for San Francisco are: Gross Margin: 21,391, Gross Margin % Dev: -1.71%, Gross Margin abs Dev: -4.28%, and Gross Margin Plan: 19,620. The values for Las Vegas are: Gross Margin: 4,778, Gross Margin % Dev: -4.43%, Gross Margin abs Dev: -6.29%, and Gross Margin Plan: 4,334. The values for Portland are: Gross Margin: 9,097, Gross Margin % Dev: 15.41%, Gross Margin abs Dev: 1.402, and Gross Margin Plan: 10,499.

6.5 Cascaded Filtering

In this example, we will explore how to do cascaded filtering; meaning filtering on dimensions and then filtering according to hierarchies (such as Flat Presentation, ABC, ...) to choose how to display the data.

We will add two Dropdown lists, one for filtering dimensions and the other for filtering hierarchies and depending on what dimension we choose to filter on, the Dropdown List for the hierarchy filters will change.

There is always one consistent filter for hierarchies which is *Flat Presentation* and according to our chosen dimension, we might either only have that one or have more options.

For example, if we are filtering on *Location*, we have two choices for hierarchies: *Flat Presentation* and according to *States*, however, if we are filtering on *Product*, we have *Flat Presentation*, *Category*, or *ABC* (this one categorizes the dimension as “worst-selling”, “medium-selling”, or “best-selling”), and if we are filtering on *Store* or *Sales Manager*, our only option is *Flat Presentation*.

The different filters can be chosen by simply selecting them from the Dropdown lists we added.

The result will look like this when we run the application:

	Gross Margin Plan	Gross Margin	Gross Margin abs Dev	Gross Margin % Dev
Los Angeles	48,542	48,972	430	0.89 %
Reno	6,898	6,083	-816	-11.83 %
Henderson	4,041	2,322	1,719	-44.96 %
Carson City	537	515	-23	-4.22 %
Portland	9,097	10,499	1,402	15.41 %
Salem	14,680	19,488	4,807	32.75 %
Eugene	7,539	8,924	1,385	18.38 %
Gresham	2,279	4,483	2,204	96.75 %
Hillsboro	537	820	283	52.56 %
Beaverton	5,797	4,088	-1,710	-29.49 %
San Francisco	21,391	19,620	-1,771	-8.28 %
San Diego	14,872	17,802	2,930	19.70 %
Sacramento	21,484	24,859	3,374	15.71 %
San Jose	27,659	24,802	-2,857	-10.33 %
Oakland	12,677	12,754	78	0.61 %
Santa Barbara	9,676	11,174	1,498	15.48 %
Beverly Hills	13,760	13,498	-262	-1.90 %
Las Vegas	4,778	4,334	-443	-9.28 %

Figure 66: Example Application Cascading Filtering

Prerequisites for this use case is having already added a Table and a Chart to your Canvas. To have all the functionalities in this use case, first go through the [Switching Between Chart and Table exercise](#).

To add a Dropdown widget for the List of Dimensions and one for the Hierarchies, we need to go to the *Insert* panel, click on the "+" icon, and choose *Dropdown*.

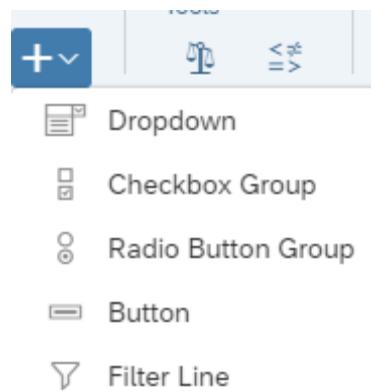
Insert two widgets into your Canvas and position them on the same level above the Table.

We will name the dropdown *Dropdown_Dimensions*.

To rename the objects, hover over them one by one and when the  icon appears click on it and choose *Rename*.

Add the second Dropdown widget for the Hierarchies we will name *Dropdown_Hierarchies*.

This use case assumes that you've a Table and a Chart already set in the Canvas. If you don't, go through the [Switching Between Chart and Table](#) and keep the names as they're in that exercise so that it works here as well.



Go into the *Builder* panel properties of the Dimensions Dropdown widget (through the Designer button on the upper right side of the screen).

Here, we will add the values that the user can choose from the widget.

Add values by clicking on the "+" icon. We will set Location to our default value.

After entering all the values, click on "Apply" to save the changes.

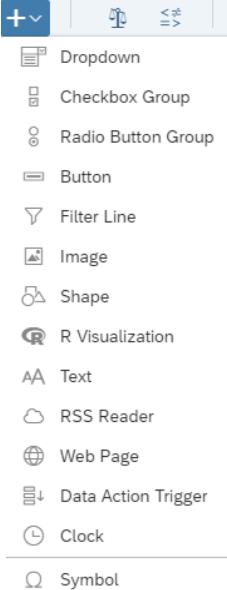
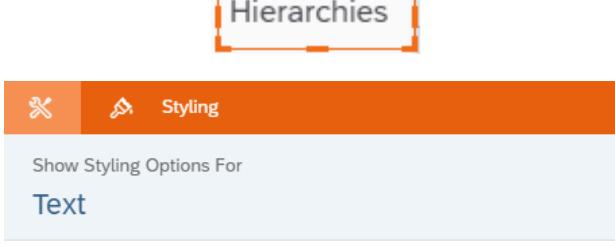
Builder

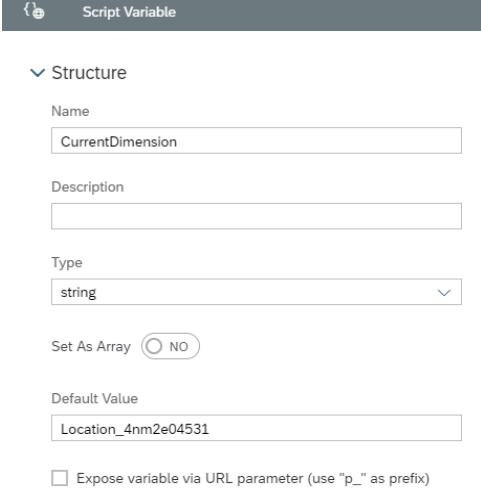
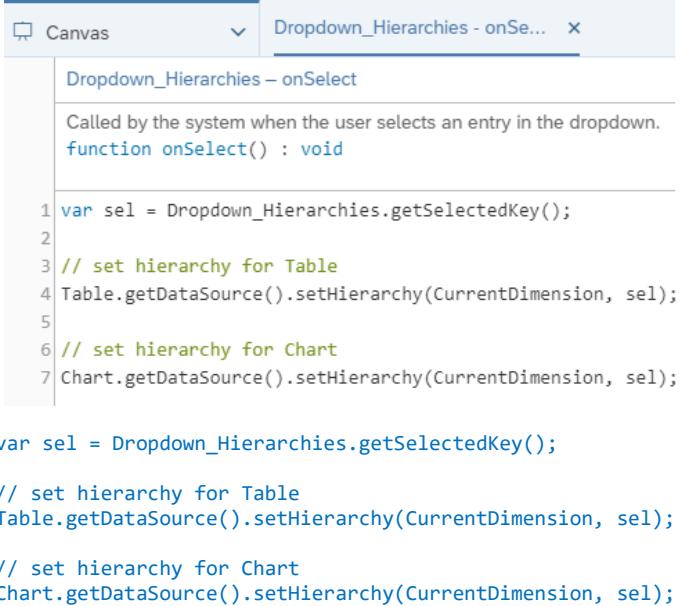
Dropdown Value

Value	Text (Optional)	Default
Location_4nm2e	Location	<input checked="" type="radio"/>
Product_3e315003an	Product	<input type="radio"/>
Store_3z2g5g06m4	Store	<input type="radio"/>
Sales_Manager_5w3m5d06b5	Sales Manager	<input type="radio"/>

Apply

Location_4nm2e04531	Location
Product_3e315003an	Product
Store_3z2g5g06m4	Store
Sales_Manager_5w3m5d06b5	Sales Manager

<p>To be able to distinguish the Dropdown List of the Dimensions and the one of the Hierarchies, we need to have labels for both.</p> <p>To add a Label, click again on the “+” icon, insert two Text widgets, and place them on the left side of each of the Dropdown Lists we added in the previous step.</p>	
<p>Now, we will set the properties of the labels we added.</p> <p>Double click on the first label and type “Dimension”</p> <p>And then go to the <i>Styling</i> panel of the label. There, we will set the name of the Label that we will use if we need to reference this widget in a script.</p> <p>Insert the name “Dropdown_Dimensions_Label”.</p>	 <p>The styling panel shows the label "Dimension" with an orange dashed border. The "Styling" tab is selected. The "Name" field contains "Dropdown_Dimensions_Label".</p>
<p>We will do the same for our second label.</p> <p>Double click on the first label and type “Hierarchies”</p> <p>And then go to the <i>Styling</i> panel of the label and</p> <p>Insert “Dropdown_Hierarchies_Label” as its Name.</p>	 <p>The styling panel shows the label "Hierarchies" with an orange dashed border. The "Styling" tab is selected. The "Name" field contains "Dropdown_Hierarchies_Label".</p>

<p>To be able to filter according to the Dimension chosen from the Dimension Dropdown list, we need to be able to store the choice in a variable that can be accessed from anywhere in the application; that means that we need a Script Variable.</p> <p>To add a script variable, click on the "+" next to SCRIPT VARIABLES that's found under Scripting.</p>	
<p>A window for the newly added script variable should now open. In the Structure part, type in "CurrentDimension" as the Name, and then set "string" as the Type and "Location_4nm2e04531" as the Default Value. This will make Location appear as our Default Value in the Dropdown widget when we run our application.</p>	
<p>To trigger the action of filtering when a choice is selected from the Dropdown Lists, we need to write an onSelect event script for them.</p> <p>We'll start with the Hierarchies Dropdown widget:</p> <p>To open the onSelect event script, hover on the Dropdown object in the Layout and click on the  icon that appears next to it.</p> <p>This script will get the selected value of the Dropdown list and accordingly set the hierarchy of the Table and the Chart while referencing our script variable, CurrentDimension, so that the hierarchy displays only correctly filtered data.</p>	 <pre> function onSelect() : void 1 var sel = Dropdown_Hierarchies.getSelectedKey(); 2 3 // set hierarchy for Table 4 Table.getDataSource().setHierarchy(CurrentDimension, sel); 5 6 // set hierarchy for Chart 7 Chart.getDataSource().setHierarchy(CurrentDimension, sel); var sel = Dropdown_Hierarchies.getSelectedKey(); // set hierarchy for Table Table.getDataSource().setHierarchy(CurrentDimension, sel); // set hierarchy for Chart Chart.getDataSource().setHierarchy(CurrentDimension, sel); </pre>

In this step, we will edit the `onSelect` event script of the Dimensions Dropdown widget:

To open the `onSelect` event script, hover on the Dropdown object in the

Layout and click on the  icon that appears next to it.

This script will get the selected choice from the Dimensions Dropdown List and save it in a variable called `sel`. The next step is to remove all the dimensions from the Table and Chart and set the selected dimension as the new dimension.

Then, from our data, we will get all the hierarchies that are available for that selected dimension, remove the hierarchies that are written now in the Hierarchies Dropdown List and loop over the available hierarchies for this selected dimension.

Lastly, we set Flat Presentation as the default hierarchy and filter our Table and Chart with the selected Dimension.

```

Canvas Dropdown_Dimensions - onSelect...
Dropdown_Dimensions - onSelect
Called by the system when the user selects an entry in the dropdown.
function onSelect() : void

1 var sel = Dropdown_Dimensions.getSelectedKey();
2
3 // Table
4 Table.removeDimension(CurrentDimension);
5 Table.addDimensionToRows(sel);
6
7 //Chart
8 Chart.removeDimension(CurrentDimension, Feed.CategoryAxis);
9 Chart.addDimension(sel, Feed.CategoryAxis);
10
11 // write filter information into the browser console
12 console.log( ['CurrentDimension: ', CurrentDimension ]);
13 console.log( ['Selection: ', sel ]);
14
15 // save the current selection (dimension) into a global variable
16 CurrentDimension = sel;
17
18 // get hierarchies from the current dimension
19 var hierarchies = Table.getDataSource().getHierarchies(CurrentDimension);
20 var flag = true;
21
22 // remove all current items form the Dropdown_Hierarchies
23 Dropdown_Hierarchies.removeAllItems();
24
25 // loop
26 for (var i=0;i<hierarchies.length; i++){
27   if (hierarchies[i].id === '__FLAT__') {
28     Dropdown_Hierarchies.addItem(hierarchies[i].id, 'Flat Presentation');
29   }
30   else {
31     Dropdown_Hierarchies.addItem(hierarchies[i].id, hierarchies[i].description);
32     if (flag === true) {
33       var hierarchy = hierarchies[i].id;
34       flag = false;
35     }
36   }
37 }
38 // write hierarchy information to browser console
39 console.log( ['Hierarchy: ', hierarchy ]);
40 console.log( ['Current Dimension: ', CurrentDimension ]);
41
42 // set Flat Hierarchy als Default
43 Dropdown_Hierarchies.setSelectedKey('__FLAT__');
44
45 // Table
46 Table.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');
47
48 // Chart
49 Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');

var sel = Dropdown_Dimensions.getSelectedKey();

// Table
Table.removeDimension(CurrentDimension);
Table.addDimensionToRows(sel);

// Chart
Chart.removeDimension(CurrentDimension, Feed.CategoryAxis);
Chart.addDimension(sel, Feed.CategoryAxis);

// write filter information into the browser console
console.log(['CurrentDimension: ', CurrentDimension]);
console.log(['Selection: ', sel]);

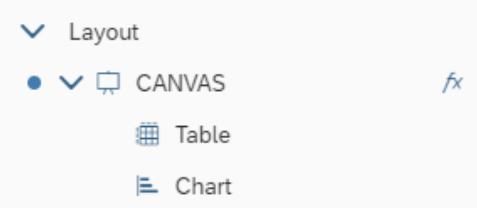
// save the current selection (dimension) into a global
variable
CurrentDimension = sel;

// get hierarchies from the current dimension
var hierarchies =
Table.getDataSource().getHierarchies(CurrentDimension);
var flag = true;

// remove all current items form the Dropdown_Hierarchies
Dropdown_Hierarchies.removeAllItems();

// loop
for (var i = 0; i < hierarchies.length; i++) {
  if (hierarchies[i].id === '__FLAT__') {
    Dropdown_Hierarchies.addItem(hierarchies[i].id, 'Flat
Presentation');
  }
  else {
}

```

	<pre>Dropdown_Hierarchies.addItem(hierarchies[i].id, hierarchies[i].description); if (flag === true) { var hierarchy = hierarchies[i].id; flag = false; } } // write hierarchy information to browser console console.log(['Hierarchy: ', hierarchy]); console.log(['Current Dimension: ', CurrentDimension]); // set Flat Hierarchie als Default Dropdown_Hierarchies.setSelectedKey('__FLAT__'); // Table Table.getDataSource().setHierarchy(CurrentDimension, '__FLAT__'); // Chart Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');</pre>
<p>The last step is setting what happens when the application is first run. This is done through the onInitialization event script of the Canvas itself.</p> <p>To get to this script, hover over the CANVAS in the Layout and click on the  icon when it appears and select <i>onInitialization</i>.</p>	

In this use case, we want to make sure that on initialization, we load all the available hierarchies of the dimensions and set Flat Presentation as the default of the Hierarchies Dropdown List.

The script for this part is the same as some of what happens when a dimension is chosen.

```

Canvas Application - onInitialization
Application - onInitialization
Called when the Analytic Application has finished loading.
function onInitialization() : void

1 // get hierarchies from the current dimension
2 var hierarchies = Table.getDataSource().getHierarchies(CurrentDimension);
3 var flag = true;
4
5 // loop
6 for (var i=0;i<hierarchies.length; i++) {
7     if (hierarchies[i].id === '__FLAT__') {
8         Dropdown_Hierarchies.addItem(hierarchies[i].id, 'Flat Presentation');
9     }
10    else {
11        Dropdown_Hierarchies.addItem(hierarchies[i].id, hierarchies[i].description);
12        if (flag === true) {
13            var hierarchy = hierarchies[i].id;
14            flag = false;
15        }
16    }
17 }
18 // write hierarchy information to browser console
19 console.log(['Hierarchy: ', hierarchy]);
20 console.log(['Current Dimension: ', CurrentDimension]);
21
22 // set Flat Hierarchie als Default
23 Dropdown_Hierarchies.setSelectedKey('__FLAT__');
24
25 //Table
26 Table.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');
27
28 //Chart
29 Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');

```

```

// get hierarchies from the current dimension
var hierarchies =
Table.getDataSource().getHierarchies(CurrentDimension);
var flag = true;

// loop
for (var i = 0; i < hierarchies.length; i++) {
    if (hierarchies[i].id === '__FLAT__') {
        Dropdown_Hierarchies.addItem(hierarchies[i].id, 'Flat
Presentation');
    }
    else {
        Dropdown_Hierarchies.addItem(hierarchies[i].id,
hierarchies[i].description);
        if (flag === true) {
            var hierarchy = hierarchies[i].id;
            flag = false;
        }
    }
}
// write hierarchy information to browser console
console.log(['Hierarchy: ', hierarchy]);
console.log(['Current Dimension: ', CurrentDimension]);

// set Flat Hierarchie als Default
Dropdown_Hierarchies.setSelectedKey('__FLAT__');

// Table
Table.getDataSource().setHierarchy(CurrentDimension,
 '__FLAT__');

// Chart
Chart.getDataSource().setHierarchy(CurrentDimension,
 '__FLAT__');

```

Now let's see how it looks like.

Click on Run Analytic Application in the upper right side of the page and the result should look something like this:

If we keep the dimension on "Location" but change the hierarchy to "States", the Table will change to display the location according to the states we have.

Now, if we change the dimension to "Product" and set the hierarchy to "Category", we will see the different categories of products displayed.

The first screenshot shows a table for 'BestRun_Advanced' with 'Location' as the dimension and 'Hierarchies' set to 'Location'. It displays data for various cities like Los Angeles, San Jose, and Las Vegas, with columns for Gross Margin Plan, Gross Margin, Gross Margin abs Dev, and Gross Margin % Dev.

The second screenshot shows the same application with 'Hierarchies' set to 'States'. It groups data by state (California, Nevada, Oregon) and provides a breakdown for each state.

The third screenshot shows the application with 'Product' as the dimension and 'Hierarchies' set to 'Category'. It displays data for product categories like Carbonated Drinks, Juices, Others, and Alcohol, with columns for Gross Margin Plan, Gross Margin, Gross Margin abs Dev, and Gross Margin % Dev.

6.6 Adding and Removing Dimensions in Rows and Columns for Table

In this example, we will, through Checkbox Groups, control which measures as well as which dimensions are displayed in the Table.

The user can select which measures they would like displayed in the Table through the Measures Checkbox and then through another Checkbox, they could decide which dimensions they want displayed on the columns or the rows of the Table.

The application also makes it easier for the user to select all or remove all measures by adding buttons specifically for that purpose.

They can also remove the dimensions that they added to the columns and rows and can choose to add them again afterwards.

The result will look like this when we run the application:

The screenshot shows the SAP Analytics Cloud interface with the title "Sample - Add and remove dimension in rows and columns for table". Below the title, a note states: "Through Checkbox Groups, we will control which measures as well as which dimensions are displayed in the Table." The interface is divided into several sections:

- Measures:** A list of measures with checkboxes: Discount abs Dev (checked), Discount % Dev (checked), and Discount (checked). Buttons include "set selected", "Remove all", and "set all".
- Dimensions:**
 - Columns:** A list with "Account" checked, with a "Remove" button.
 - Rows:** A list with "Product" checked, with a "Remove" button.
 - Free:** A list with "Currency", "Time", "Location", and "Sales Manager" unselected, with "add to Column" and "add to Row" buttons.
- Table:** A data grid titled "BestRun_Advanced" with the subtitle "in Thousand". The table has 16 columns and 21 rows, including a header row. The columns are: Discount abs Dev, Discount % Dev, Discount, Discount Plan, Gross Margin abs Dev, Gross Margin % Dev, Gross Margin, Gross Margin Plan, and Original Sale.

Figure 67: Add and Remove Dimensions

This application assumes that there already is a Table in your Canvas. To match the scripts in the application we recommend renaming the widget to [Table](#).

We will start by adding 5 Checkbox Groups.
 The first one will display all the available measures and the user can choose which ones they want to see in the Table, the second one will display the dimensions we want our Columns to be filtered on, while the third does the same but for our Rows.
 The fourth Checkbox Group will display the dimensions that we could add to the second and third Checkbox.

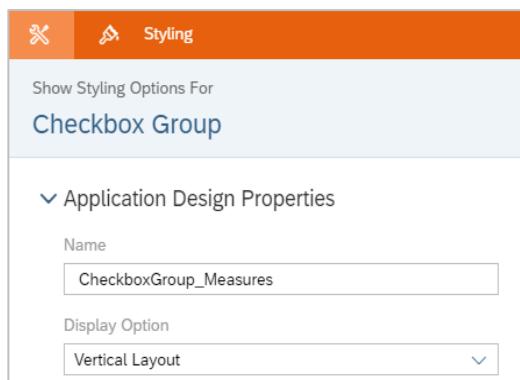
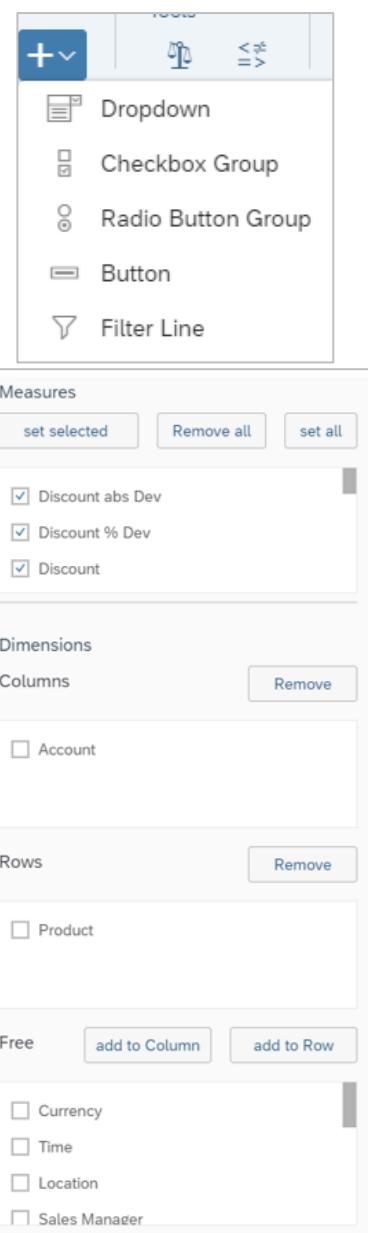
Place the first four Checkbox Groups under each other on the left side of the Table. (as shown in the screenshot).

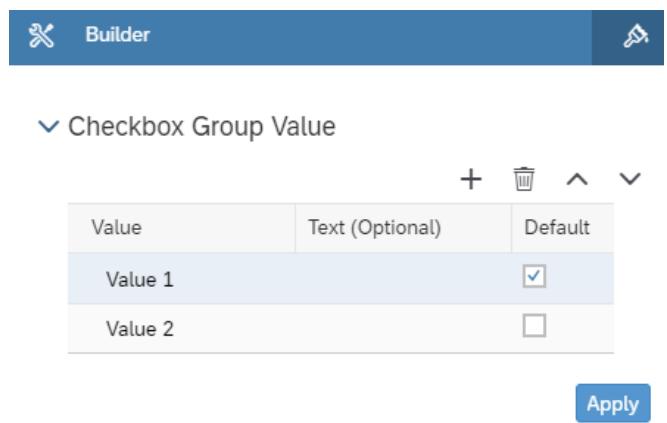
The fifth Checkbox Group will get the selected dimensions of the fourth Checkbox and order the Checkboxes according to the selections while also taking care that there aren't any repetitions in any of the other Checkbox Groups.

To start off, click on the "+" icon in the *Insert* panel and choose *Checkbox Group* and place on the left side of the Table.

Go to the Designer of the first Checkbox (by clicking on Designer on the upper right side of the screen) and switch to the *Styling* panel by clicking on the  button.

There, enter "CheckboxGroup_Measures" as the Name and choose *Vertical Layout* as the Display Option.



<p>Switch over to the <i>Builder</i> panel of the same widget and delete the values in the Table. Simply select the value and click on the  icon to delete it.</p> <p>Afterwards, click on Apply to save the changes.</p>	 <p>The screenshot shows the 'Builder' panel for a 'Checkbox Group Value' configuration. At the top, there's a toolbar with a trash can icon, a magnifying glass icon, and other controls. Below the toolbar is a section titled 'Checkbox Group Value' with a dropdown arrow. A table below lists two items:</p> <table border="1" data-bbox="774 422 1319 563"><thead><tr><th>Value</th><th>Text (Optional)</th><th>Default</th></tr></thead><tbody><tr><td>Value 1</td><td></td><td><input checked="" type="checkbox"/></td></tr><tr><td>Value 2</td><td></td><td><input type="checkbox"/></td></tr></tbody></table> <p>At the bottom right of the panel is a blue 'Apply' button.</p>	Value	Text (Optional)	Default	Value 1		<input checked="" type="checkbox"/>	Value 2		<input type="checkbox"/>
Value	Text (Optional)	Default								
Value 1		<input checked="" type="checkbox"/>								
Value 2		<input type="checkbox"/>								

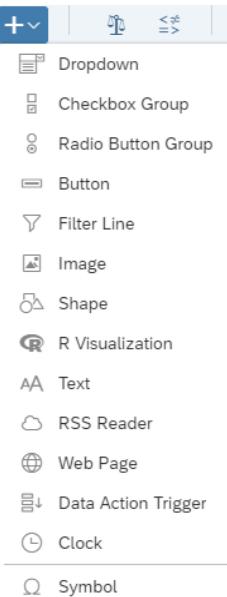
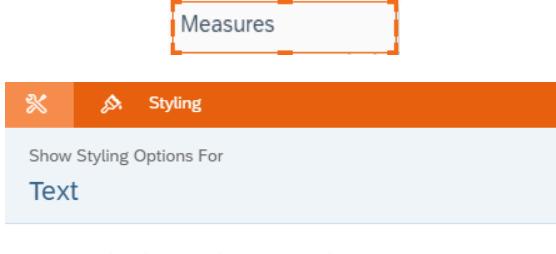
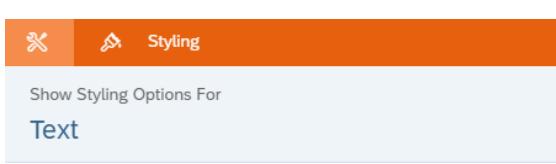
We will do the same for the other Checkbox Groups. Add four new Checkbox Groups; for the first enter "CheckboxGroup_Columns", for the second enter "CheckboxGroup_Rows", for the third enter "CheckboxGroup_Free", and for the last enter "CheckboxGroup_AllDimensions" as the Name. Choose "Vertical Layout" as the Display Option for all of them.

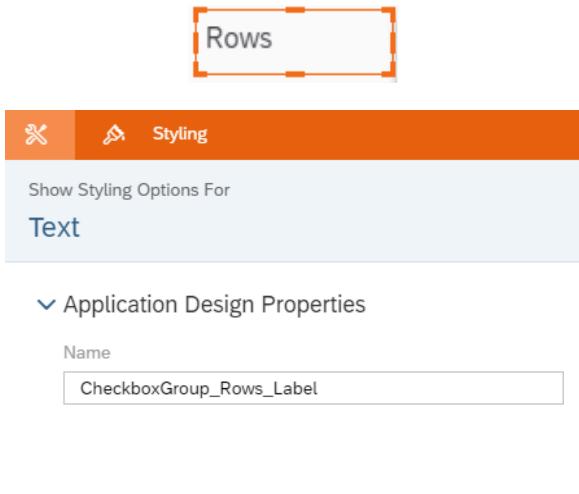
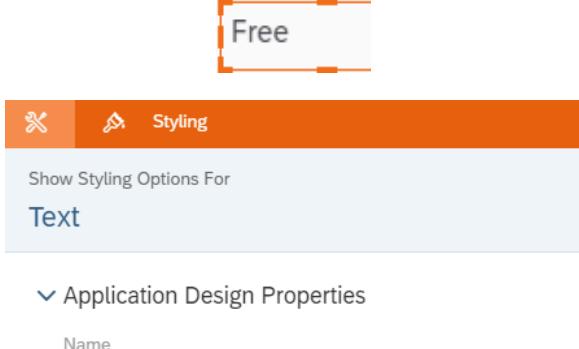
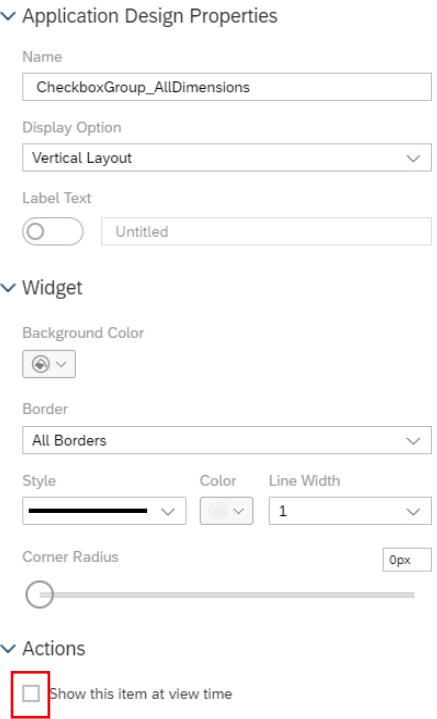
Place the three Checkbox Groups under each other on the left side of the Table, under the Measure Checkbox, in the order in which we inserted them (as described before).

Place the last checkbox as indicated in the screenshot on the right or somewhere in the Canvas (the place isn't important because the checkbox will be hidden).

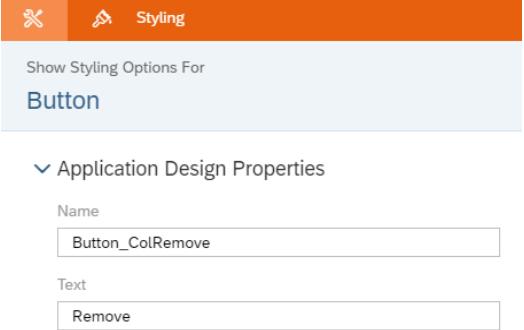
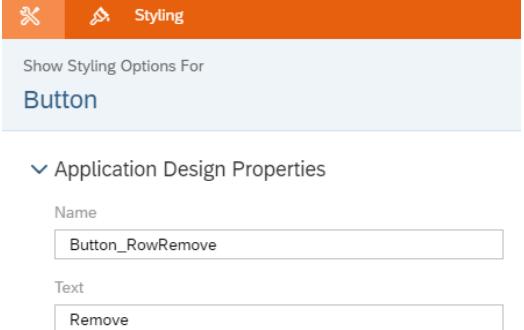
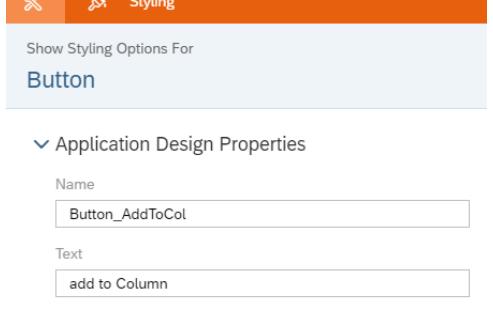
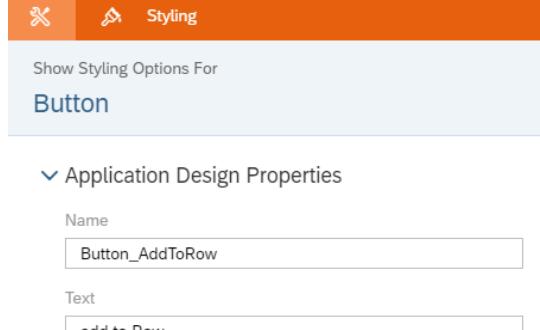
After editing these values, go to the *Builder* panel of each of the Checkbox Groups and delete the values that are there like we did in the first Checkbox Group.

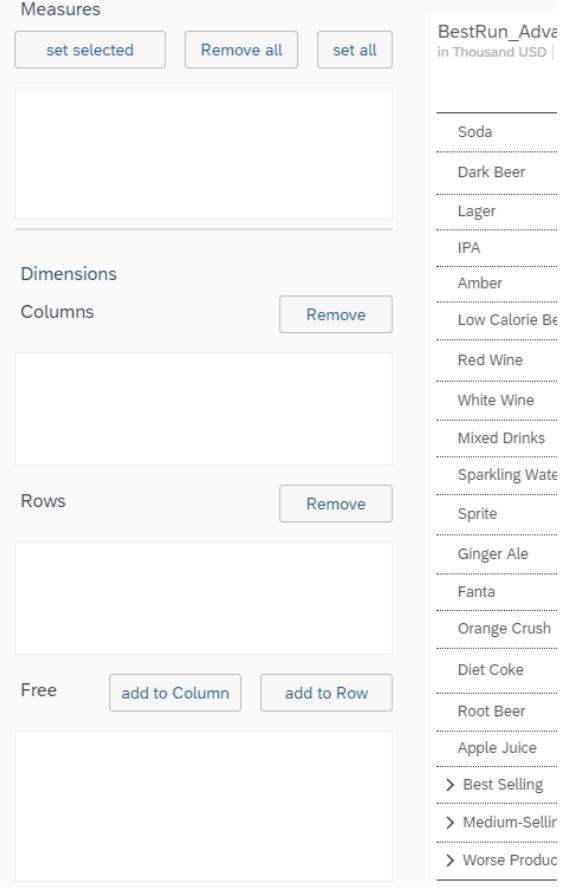
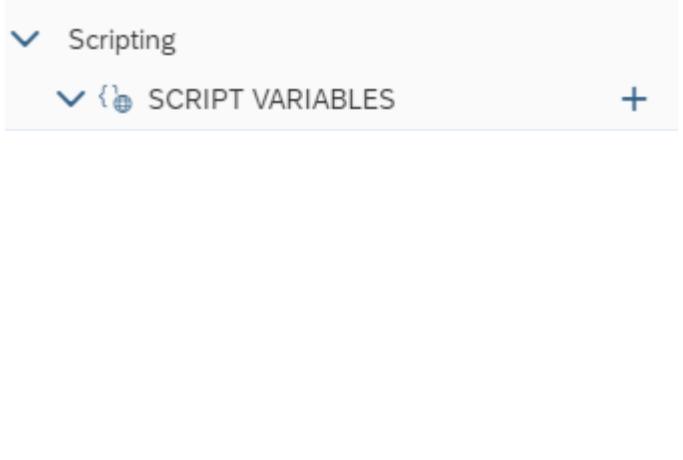


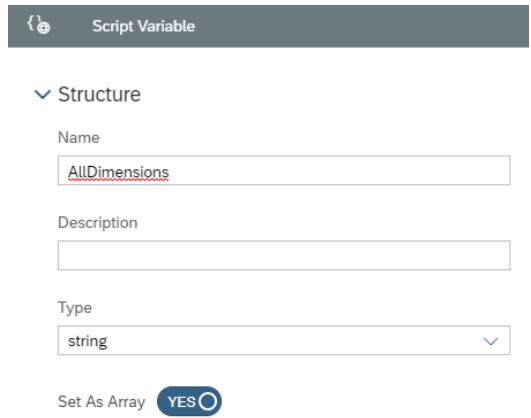
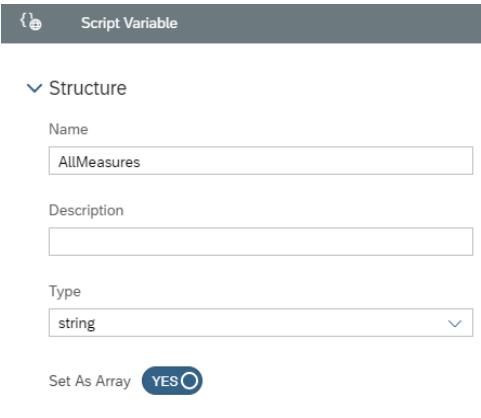
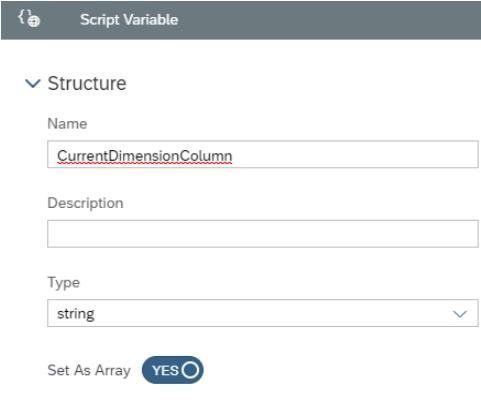
<p>To be able to distinguish the Checkbox Groups from each other, we need to have labels for four of them. (We don't need a label for the All Dimensions Checkbox because it won't be visible at view time)</p> <p>To add a Label, click again on the "+" icon, insert four Text widgets, and place each one of them above each of the Checkbox Groups we added.</p>	
<p>Now, we will set the properties of the labels we added. Double click on the first label and type "Measures". And then go to the <i>Styling</i> panel of the label. There, we will set the name of the Label that we will use if we need to reference this widget in a script. Insert the name "CheckboxGroup_Measures_Label".</p>	 <p>Measures</p> <p>Styling</p> <p>Show Styling Options For Text</p> <p>Application Design Properties</p> <p>Name: CheckboxGroup_Measures_Label</p>
<p>We will do the same for our second label. Double click on the first label and type "Columns". And then go to the <i>Styling</i> panel of the label and insert "CheckboxGroup_Columns_Label" in its Name field.</p>	 <p>Columns</p> <p>Styling</p> <p>Show Styling Options For Text</p> <p>Application Design Properties</p> <p>Name: CheckboxGroup_Columns_Label</p>

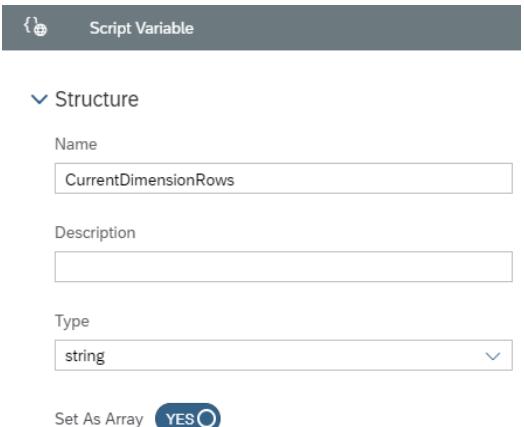
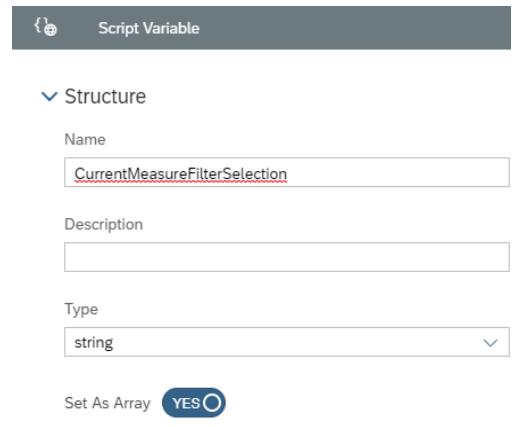
<p>Navigate towards the third label and there: Double click on the first label and type "Rows" And then go to the <i>Styling</i> panel of the label and Insert the Name "CheckboxGroup_Rows_Label".</p>	
<p>Finally, we will edit our fourth label. Double click on the first label and type "Free" And then go to the <i>Styling</i> panel of the label and insert the name "CheckboxGroup_Free_Label" as its Name.</p>	
<p>Now, we need to set the AllDimensions Checkbox to invisible at view time because we only need it to sort our dimensions as you'll see later in the exercise. Go into the <i>Styling</i> panel of the CheckboxGroup_AllDimensions and uncheck Show this item at view time</p>	

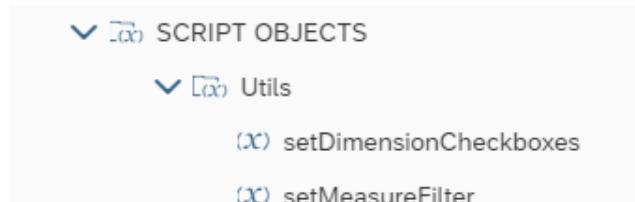
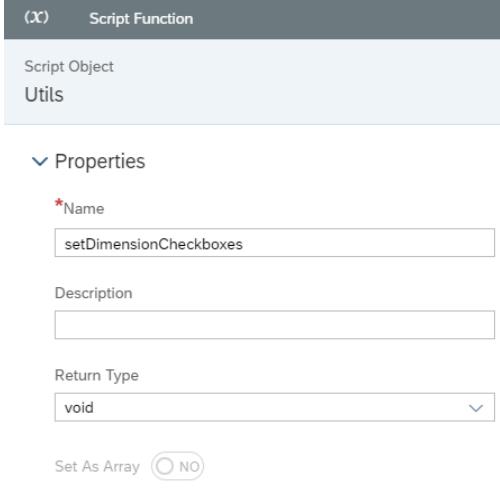
<p>Now, we will add all the buttons, we need, to control our choices from the Checkbox Groups.</p> <p>To add our first button, click on the "+" icon and insert a Button and place it between the Measures Label and its Checkbox Group.</p> <p>To edit the button, click on it and go to the <i>Styling</i> panel.</p> <p>For the Name, enter "Button_setMeasureFilter" and for the Text enter "set selected". This button will set the measures we choose from the Measures Checkbox as measures for our Table.</p>	 <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Dropdown <input type="checkbox"/> Checkbox Group <input type="checkbox"/> Radio Button Group <input type="checkbox"/> Button <input type="checkbox"/> Filter Line <input type="checkbox"/> Image <input type="checkbox"/> Shape <div style="background-color: #f08030; color: white; padding: 5px; margin-top: 10px;"> ✖ ✖ Styling </div> <p>Show Styling Options For Button</p> <p>Application Design Properties</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Name</td> <td style="padding: 2px;"><input type="text" value="Button_setMeasureFilter"/></td> </tr> <tr> <td style="padding: 2px;">Text</td> <td style="padding: 2px;"><input type="text" value="set selected"/></td> </tr> </table>	Name	<input type="text" value="Button_setMeasureFilter"/>	Text	<input type="text" value="set selected"/>
Name	<input type="text" value="Button_setMeasureFilter"/>				
Text	<input type="text" value="set selected"/>				
<p>Now, we will do the same for all the buttons we need.</p> <p>Add a new button and place it next to the first one.</p> <p>For this button, enter "Button_removeAllMeasures" for the Name and "Remove all" for the Text. This button will be used to uncheck all the measures from the Measures Checkbox Group and set the measure filters for the Table to empty.</p>	<div style="background-color: #f08030; color: white; padding: 5px; margin-top: 10px;"> ✖ ✖ Styling </div> <p>Show Styling Options For Button</p> <p>Application Design Properties</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Name</td> <td style="padding: 2px;"><input type="text" value="Button_removeAllMeasures"/></td> </tr> <tr> <td style="padding: 2px;">Text</td> <td style="padding: 2px;"><input type="text" value="Remove all"/></td> </tr> </table>	Name	<input type="text" value="Button_removeAllMeasures"/>	Text	<input type="text" value="Remove all"/>
Name	<input type="text" value="Button_removeAllMeasures"/>				
Text	<input type="text" value="Remove all"/>				
<p>Add a third button and place it next to the second one.</p> <p>For this button, enter "Button_setAllMeasures" for the Name and "set all" for the Text. This button will be used to set all the available measures as measures for our Table.</p>	<div style="background-color: #f08030; color: white; padding: 5px; margin-top: 10px;"> ✖ ✖ Styling </div> <p>Show Styling Options For Button</p> <p>Application Design Properties</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Name</td> <td style="padding: 2px;"><input type="text" value="Button_setAllMeasures"/></td> </tr> <tr> <td style="padding: 2px;">Text</td> <td style="padding: 2px;"><input type="text" value="set all"/></td> </tr> </table>	Name	<input type="text" value="Button_setAllMeasures"/>	Text	<input type="text" value="set all"/>
Name	<input type="text" value="Button_setAllMeasures"/>				
Text	<input type="text" value="set all"/>				

<p>Add a new button and place it next to the “Columns” Label. This button’s Name will be set to “Button_ColRemove” and its Text will read “Remove” and it will be used to remove the dimensions that the user selects in the Columns Checkbox from the Checkbox as well as from the columns of our Table.</p>	 <p>Show Styling Options For Button</p> <p>Application Design Properties</p> <p>Name Button_ColRemove</p> <p>Text Remove</p>
<p>Next to the Rows Label, insert a new button and enter the Name “Button_RowRemove” and the Text “Remove” in its properties’ settings. This button will be used to remove the dimensions that the user selects in the Rows Checkbox from the Checkbox as well as the rows of our Table.</p>	 <p>Show Styling Options For Button</p> <p>Application Design Properties</p> <p>Name Button_RowRemove</p> <p>Text Remove</p>
<p>Next to the Free Label we will add two buttons; for the first, insert a new button and enter the Name “Button_AddToCol” and the Text “add to Column” in its properties’ settings. When this button is clicked, the selected dimensions from the Free Checkbox Group will be added as dimensions of the Table’s Columns.</p>	 <p>Show Styling Options For Button</p> <p>Application Design Properties</p> <p>Name Button_AddToCol</p> <p>Text add to Column</p>
<p>Next to the previous button, add another button and enter “Button_AddToRow” for the Name and “add to Row” for the Text. When this button is clicked, the selected dimensions from the Free Checkbox Group will be added as dimensions to the Rows in the Table.</p>	 <p>Show Styling Options For Button</p> <p>Application Design Properties</p> <p>Name Button_AddToRow</p> <p>Text add to Row</p>

<p>Compare your Canvas to the screenshot on the right and make sure they look alike.</p> <p>We won't add a label for the last Checkbox Group (All Dimensions) since it's there to simply help us set the dimensions in the Columns, Rows, and Free Checkboxes so that there are no repetitions.</p>	 <p>Measures</p> <p>set selected Remove all set all</p> <p>Dimensions</p> <p>Columns Remove</p> <p>Rows Remove</p> <p>Free add to Column add to Row</p> <p>BestRun_Adve in Thousand USD </p> <ul style="list-style-type: none"> Soda Dark Beer Lager IPA Amber Low Calorie Be Red Wine White Wine Mixed Drinks Sparkling Wate Sprite Ginger Ale Fanta Orange Crush Diet Coke Root Beer Apple Juice > Best Selling > Medium-Sellin > Worse Produc
<p>To be able to filter according to the measures and dimensions chosen from the Checkbox Groups, we need to be able to store the choices in variables that can be accessed from anywhere in the application; that means that we need Script Variables.</p> <p>To add a script variable, click on the "+" next to SCRIPT VARIABLES that's under Scripting.</p>	 <p>Scripting</p> <p>SCRIPT VARIABLES +</p>

<p>A window for the newly added script variable should now open. In the Structure part, type in "AllDimensions" as the Name, and then set "string" as the Type and toggle the Set As Array button to Yes.</p> <p>This variable will hold all the dimensions in our data set.</p>	 <p>The screenshot shows the 'Script Variable' dialog. At the top, there's a dark header bar with the title 'Script Variable'. Below it, a section titled 'Structure' is expanded. The 'Name' field contains 'AllDimensions'. The 'Type' dropdown is set to 'string'. A 'Set As Array' button is shown as 'YES' with a blue outline. The 'Description' field is empty.</p>
<p>Now, we will add a second script variable that will hold all the measures of our data set. Add a new variable like we did in the previous 2 steps. In the name field insert "AllMeasures", set the Type to "string", and toggle the Set As Array button to Yes.</p>	 <p>The screenshot shows the 'Script Variable' dialog. At the top, there's a dark header bar with the title 'Script Variable'. Below it, a section titled 'Structure' is expanded. The 'Name' field contains 'AllMeasures'. The 'Type' dropdown is set to 'string'. A 'Set As Array' button is shown as 'YES' with a blue outline. The 'Description' field is empty.</p>
<p>To be able to implement the selected dimensions in our Columns and Rows, we need to save these in a script variable. Firstly, we will insert a script variable to hold the selected dimensions that we have chosen to add to our Columns. Add a new script variable and enter "CurrentDimensionColumn" in the Name field, set string as Type, and toggle the Set As Array button to Yes.</p>	 <p>The screenshot shows the 'Script Variable' dialog. At the top, there's a dark header bar with the title 'Script Variable'. Below it, a section titled 'Structure' is expanded. The 'Name' field contains 'CurrentDimensionColumn'. The 'Type' dropdown is set to 'string'. A 'Set As Array' button is shown as 'YES' with a blue outline. The 'Description' field is empty.</p>

<p>To hold the selected dimensions, we have chosen to add to our Rows, we will insert a new script variable. Type "CurrentDimensionRows" in the Name field, set the Type to string, and toggle the Set As Array button to Yes.</p>	 <p>The screenshot shows the "Script Variable" configuration dialog. The "Name" field contains "CurrentDimensionRows". The "Type" dropdown is set to "string". The "Set As Array" button is labeled "YES".</p>
<p>Our final script variable will hold the measure(s) we have selected from the Measures Checkbox Group. Insert a new script variable and set the Name to "CurrentMeasureFilterSelection", the Type to string, and the Set As Array to Yes.</p>	 <p>The screenshot shows the "Script Variable" configuration dialog. The "Name" field contains "CurrentMeasureFilterSelection". The "Type" dropdown is set to "string". The "Set As Array" button is labeled "YES".</p>

<p>To define what should happen when a dimension or a measure is chosen, we need to create a Script Object. In this object, we will create a function that sets the measure filter according to what the user has chosen from the Measures Checkbox Group and another function that sets the dimensions according to what the user has chosen from the Free Checkbox Group.</p> <p>To create a Script Object, select the “+” icon next to SCRIPT OBJECTS under the Layout.</p> <p>This will add only one script function to the script object.</p> <p>To add a second one, hover over the folder created, click on the  icon when it appears and click on “+ Add Script Function”.</p> <p>Rename all the added elements as the following: We will name the folder Utils, the first function setDimensionCheckboxes and the second function setMeasureFilter.</p> <p>To rename the objects, hover over them one by one and when the  icon appears click on it and choose <i>Rename</i>.</p>	
<p>Click on the function setDimensionCheckboxes and set the Return Type to void.</p>	

Click on the function setMeasureFilter and when the Properties window opens, set the Return Type to void and click on the "+" icon next to Arguments. There, add an argument with the name "selectedIds" and the type string[] (string array).

The screenshot shows a software interface for defining a script function. The top section is titled '(X) Script Function'. It lists 'Script Object' and 'Utils' under 'Category'. Below this, the 'Properties' section is expanded, showing:

- Name:** setMeasureFilter
- Description:** (empty)
- Return Type:** void
- Set As Array:** NO (radio button)

The 'Arguments' section is expanded, showing one argument:

- Argument:** selectedIds
- Script Function:** Utils – setMeasureFilter

The 'Settings' section is expanded, showing:

- Name:** selectedIds
- Type:** string
- Set As Array:** YES (radio button)

Now, we can write the script for the functions.

 Click on the  icon next to the `setDimensionCheckboxes` function. Here, we will define what happens when a user selects dimensions from the Free Checkbox Group to be added to the Columns or the Rows.

Firstly, we will remove all items from the Column, Rows, and Free Checkboxes.

Then, we will call on the `create` function of the script variables and create two new string arrays and save one in our `CurrentDimensionColumn` script variable and the other in the `CurrentDimensionRows` script variable.

Afterwards, we get the dimensions that are now on the Table's columns and push each on into the string array of `CurrentDimensionColumn`. We then do the same for the Rows, this time pushing the row dimensions into the string array of `CurrentDimensionRows`.

We then get all the dimensions, and we will see which dimensions were chosen from the `AllDimensions` checkbox.

Next, we will add these dimensions to our Free Checkbox but remove the ones that are in the Rows or Columns Checkboxes so that we don't have any repetitions between the three Checkboxes.

```
Canvas Utils - setDimensionCheckboxes
Utils - setDimensionCheckboxes

function setDimensionCheckboxes() : void

1 CheckboxGroup_Columns.removeAllItems();
2 CheckboxGroup_Rows.removeAllItems();
3 CheckboxGroup_Free.removeAllItems();
4

5
6 CurrentDimensionColumn = ArrayUtils.create(Type.string);
7 CurrentDimensionRows = ArrayUtils.create(Type.string);
8 console.log(["CurrentDimensionColumn should empty", CurrentDimensionColumn.slice()]);
9 console.log(["CurrentDimensionRows should empty", CurrentDimensionRows.slice()]);
10
11 // Dimension in Columns
12 var dimCol = Table.getDimensionsOnColumns();
13 if (dimCol.length > 0) {
14     for (var i=0;i<dimCol.length; i++){
15         CurrentDimensionColumn.push(dimCol[i]);
16         console.log(["CurrentDimensionColumn ", dimCol[i]]);
17     }
18 }
19
20
21 // Dimension in Rows
22 var dimRows = Table.getDimensionsOnRows();
23 if (dimRows.length > 0) {
24     for (i=0;i<dimRows.length; i++){
25         CurrentDimensionRows.push(dimRows[i]);
26         console.log(["CurrentDimensionRows ", dimRows[i]]);
27     }
28 }
29
30 // get all Dimensions
31 if (AllDimensions.length > 0) {
32     for (i=0;i<AllDimensions.length; i++){
33         if (AllDimensions[i] != "") {
34             CheckboxGroup_AllDimensions.setSelectedKeys([AllDimensions[i]]);
35             var dimdesc = CheckboxGroup_AllDimensions.getSelectedTexts();
36             CheckboxGroup_Free.addItem(AllDimensions[i],dimdesc[0]);
37             console.log(["AllDimensions",AllDimensions[i], dimdesc[0]]);
38         }
39     }
40 }
41
42 console.log(["CurrentDimensionColumn", CurrentDimensionColumn]);
43 console.log(["CurrentDimensionRows", CurrentDimensionRows]);
44
45 // remove the dimensions from the free list, which are in rows / columns
46 if (CurrentDimensionRows.length > 0) {
47     for (i=0;i<CurrentDimensionRows.length; i++){
48         if (CurrentDimensionRows[i] != "") {
49             CheckboxGroup_Free.setSelectedKeys([CurrentDimensionRows[i]]);
50             dimdesc = CheckboxGroup_Free.getSelectedTexts();
51             CheckboxGroup_Rows.addItem(CurrentDimensionRows[i],dimdesc[0]);
52             CheckboxGroup_Free.removeItem(CurrentDimensionRows[i]);
53         }
54     }
55 }
56
57 if (CurrentDimensionColumn.length > 0) {
58     for (i=0;i<CurrentDimensionColumn.length; i++){
59         if (CurrentDimensionColumn[i] != "") {
60             CheckboxGroup_Free.setSelectedKeys([CurrentDimensionColumn[i]]);
61             dimdesc = CheckboxGroup_Free.getSelectedTexts();
62             CheckboxGroup_Columns.addItem(CurrentDimensionColumn[i],dimdesc[0]);
63             CheckboxGroup_Free.removeItem(CurrentDimensionColumn[i]);
64         }
65     }
66 }
67

CheckboxGroup_Columns.removeAllItems();
CheckboxGroup_Rows.removeAllItems();
CheckboxGroup_Free.removeAllItems();

CurrentDimensionColumn = ArrayUtils.create(Type.string);
CurrentDimensionRows = ArrayUtils.create(Type.string);
console.log(["CurrentDimensionColumn should empty",
CurrentDimensionColumn.slice()]);
console.log(["CurrentDimensionRows should empty",
CurrentDimensionRows.slice()]);

// Dimension in Columns
var dimCol = Table.getDimensionsOnColumns();
if (dimCol.length > 0) {
    for (var i = 0; i < dimCol.length; i++) {
        CurrentDimensionColumn.push(dimCol[i]);
        console.log(["CurrentDimensionColumn ", dimCol[i]]);
    }
}
```

```
// Dimension in Rows
var dimRows = Table.getDimensionsOnRows();
if (dimRows.length > 0) {
    for (i = 0; i < dimRows.length; i++) {
        CurrentDimensionRows.push(dimRows[i]);
        console.log(["CurrentDimensionRows ", dimRows[i]]);
    }
}

// get all Dimensions
if (AllDimensions.length > 0) {
    for (i = 0; i < AllDimensions.length; i++) {
        if (AllDimensions[i] != "") {

CheckboxGroup_AllDimensions.setSelectedKeys([AllDimensions[i]]);
        var dimdesc =
CheckboxGroup_AllDimensions.getSelectedTexts();
        CheckboxGroup_Free.addItem(AllDimensions[i],
dimdesc[0]);
        console.log(["AllDimensions",AllDimensions[i],
dimdesc[0]]);
    }
}
}

console.log(["CurrentDimensionColumn",
CurrentDimensionColumn]);
console.log(["CurrentDimensionRows",
CurrentDimensionRows]);

// remove the dimensions from the free list, which are in
// rows/columns
if (CurrentDimensionRows.length > 0) {
    for (i = 0; i < CurrentDimensionRows.length; i++) {
        if (CurrentDimensionRows[i] != "") {

CheckboxGroup_Free.setSelectedKeys([CurrentDimensionRows[i]]);
        dimdesc = CheckboxGroup_Free.getSelectedTexts();
        CheckboxGroup_Rows.addItem(CurrentDimensionRows[i],
dimdesc[0]);

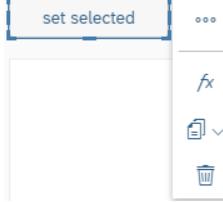
CheckboxGroup_Free.removeItem(CurrentDimensionRows[i]);
    }
}
}

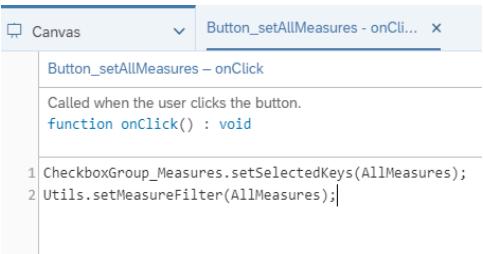
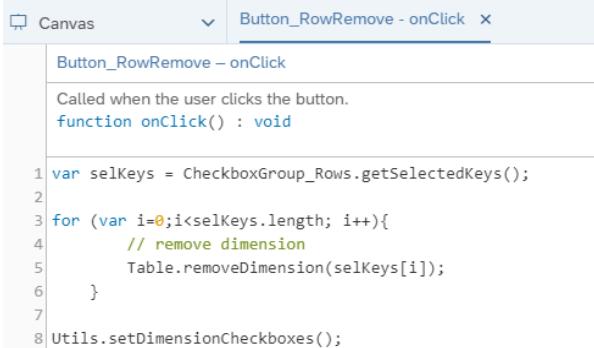
if (CurrentDimensionColumn.length > 0) {
    for (i = 0; i < CurrentDimensionColumn.length; i++) {
        if (CurrentDimensionColumn[i] != "") {

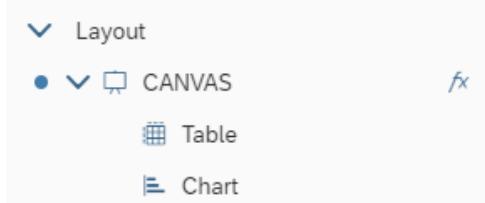
CheckboxGroup_Free.setSelectedKeys([CurrentDimensionColumn[i]]);
        dimdesc = CheckboxGroup_Free.getSelectedTexts();

CheckboxGroup_Columns.addItem(CurrentDimensionColumn[i],
dimdesc[0]);

CheckboxGroup_Free.removeItem(CurrentDimensionColumn[i]);
    }
}
}
```

<p>Now, we will do the same for the setMeasureFilter function. Click on the  icon next to the setMeasureFilter function and there, we will define what happens to the Table when a user selects a measure from the Dropdown list.</p> <p>We will remove any already set dimension filter of the Table and then we will add the selectedIds as the new dimension(s) of the Table.</p> <p>Finally, we will save the selected measures in our CurrentMeasureFilterSelection script variable.</p>	<pre>Canvas Utils - setMeasureFilter function setMeasureFilter(selectedIds: string[]): void 1 // remove Measures 2 Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold"); 3 4 // add Measures 5 Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold", selectedIds); 6 7 // save the current selection into global variable 8 CurrentMeasureFilterSelection = selectedIds; // remove Measures Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold"); // add Measures Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold", selectedIds); // save the current selection into global variable CurrentMeasureFilterSelection = selectedIds;</pre>
<p>To trigger an action when our buttons are clicked, we need to write <code>onClick</code> event scripts for them.</p> <p>Let's start with the first button, setMeasureFilter (Text: set selected). Click on the button in your Canvas and select the  icon.</p>	
<p>In the script of this button, we will get the selected keys of the Measures Checkbox and using the function <code>Utils.setMeasureFilter</code>, we will set them as the measure filters for our table.</p>	<pre>Canvas Button_setMeasureFilter - onClick Called when the user clicks the button. function onClick(): void 1 Utils.setMeasureFilter(CheckboxGroup_Measures.getSelectedKeys()); Utils.setMeasureFilter(CheckboxGroup_Measures.getSelectedKeys());</pre>
<p>Next, we will edit the <code>onClick</code> event script of the second button, removeAllMeasures (Text: Remove All). Click on the button in your Canvas and select the  icon. Here, we will set the selected keys and the measure filter to empty arrays.</p>	<pre>Canvas Button_removeAllMeasures - onClick Called when the user clicks the button. function onClick(): void 1 CheckboxGroup_Measures.setSelectedKeys([""]); 2 Utils.setMeasureFilter([""]); CheckboxGroup_Measures.setSelectedKeys([""]); Utils.setMeasureFilter([""]);</pre>

<p>The script of the third button, Button_setAllMeasures (Text: set all), will set the selected keys of the Checkbox Group to the script variable AllMeasures and use the Utils.setMeasureFilter function to set the measure filter to all measures.</p>	 <pre>Button_setAllMeasures - onClick Called when the user clicks the button. function onClick() : void 1 CheckboxGroup_Measures.setSelectedKeys(AllMeasures); 2 Utils.setMeasureFilter(AllMeasures);</pre> <p><code>CheckboxGroup_Measures.setSelectedKeys(AllMeasures); Utils.setMeasureFilter(AllMeasures);</code></p>
<p>The fourth button's script, Button_ColRemove (Text: Remove), when triggered, gets the selected keys of the Columns Checkbox Group and then removes these dimensions from the Table and then calls the setDimensionCheckboxes function to set the Checkboxes according to the new selections.</p>	 <pre>Button_ColRemove - onClick Called when the user clicks the button. function onClick() : void 1 var selKeys = CheckboxGroup_Columns.getSelectedKeys(); 2 3 for (var i=0;i<selKeys.length; i++){ 4 // remove dimension 5 Table.removeDimension(selKeys[i]); 6 } 7 8 Utils.setDimensionCheckboxes();</pre> <p><code>var selKeys = CheckboxGroup_Columns.getSelectedKeys(); for (var i = 0; i < selKeys.length; i++) { // remove dimension Table.removeDimension(selKeys[i]); } Utils.setDimensionCheckboxes();</code></p>
<p>Now, we will edit the script of the button Button_RowRemove (Text: Remove). Here, we will do the same as in step 32 with the ColRemove button. We will get the selected keys of the Rows Checkbox Group and then remove these dimensions from the Table and call the setDimensionCheckboxes function to reset the checkboxes again according to the new selections.</p>	 <pre>Button_RowRemove - onClick Called when the user clicks the button. function onClick() : void 1 var selKeys = CheckboxGroup_Rows.getSelectedKeys(); 2 3 for (var i=0;i<selKeys.length; i++){ 4 // remove dimension 5 Table.removeDimension(selKeys[i]); 6 } 7 8 Utils.setDimensionCheckboxes();</pre> <p><code>var selKeys = CheckboxGroup_Rows.getSelectedKeys(); for (var i = 0; i < selKeys.length; i++) { // remove dimension Table.removeDimension(selKeys[i]); } Utils.setDimensionCheckboxes();</code></p>

<p>The fifth button, Button_AddToCol (Text: add to Column) will, when clicked on, get the selected keys of the Free Checkbox and add the dimensions to the column of the Table.</p> <p>The script will then call the setDimensionCheckboxes function to set the Checkboxes to the new selection.</p>	<pre>Canvas Button_AddToCol - onClick Button_AddToCol – onClick Called when the user clicks the button. function onClick() : void 1 var selKeys = CheckboxGroup_Free.getSelectedKeys(); 2 3 for (var i=0;i<selKeys.length; i++){ 4 // add dimension to Column in table 5 Table.addDimensionToColumns(selKeys[i]); 6 } 7 8 Utils.setDimensionCheckboxes(); var selKeys = CheckboxGroup_Free.getSelectedKeys(); for (var i = 0; i < selKeys.length; i++) { // add dimension to Column in table Table.addDimensionToColumns(selKeys[i]); } Utils.setDimensionCheckboxes();</pre>
<p>The script of the last button, Button_AddRow (Text: add to Row), will get the selected keys of the Free Checkbox and add the dimensions to the Rows of the Table, and then, same as the previous script, it will call the setDimensionCheckboxes function to set the Checkboxes to the new selection.</p>	<pre>Canvas Button_AddRow - onClick Button_AddRow – onClick Called when the user clicks the button. function onClick() : void 1 var selKeys = CheckboxGroup_Free.getSelectedKeys(); 2 3 for (var i=0;i<selKeys.length; i++){ 4 // remove dimension 5 Table.addDimensionToRows(selKeys[i]); 6 } 7 8 Utils.setDimensionCheckboxes(); var selKeys = CheckboxGroup_Free.getSelectedKeys(); for (var i = 0; i < selKeys.length; i++) { // remove dimension Table.addDimensionToRows(selKeys[i]); } Utils.setDimensionCheckboxes();</pre>
<p>The last step is deciding what happens when the application is first run.</p> <p>This is done through the onInitialization event script of the Canvas itself.</p> <p>To get to this script, hover over the CANVAS in the Layout and click on the  icon when it appears.</p>	

In this use case, we want to make sure that on initialization, we get all the measures from the data source of the Table.

We will then define an array of type string and call it selectedKeys. Afterwards, we will add all the measures to the Measures Checkbox Group as well as the selectedKeys array.

We will then set the selected keys of the Checkbox Group to the selectedKeys variable and set our script variable AllMeasures to selectedKeys since it still holds all the measures of our data set.

Afterwards, we define another string array and put all the dimensions of the data source in it as well as add these dimensions as items of the Checkbox Group of all dimensions (CheckboxGroup_AllDimensions).

Next, we will set the script variable AllDimensions to the string array (selectedDims) that we have created to store the dimensions in.

The last step is to call the functions of setMeasureFilter to set the selected keys to the array we had defined at the beginning (selectedKeys) and to call the setDimensionCheckboxes function to set the dimension checkboxes to its initial state.

```
Application - onInitialization
Called when the Analytic Application has finished loading.
function onInitialization() : void

1 // Measures
2 // get all measures from the table data source
3 var measures = Table.getDataSource().getMeasures();
4
5 // define array or the electe
6 var selectedKeys = ArrayUtils.create(Type.string);
7
8 if (measures.length > 0) {
9     for (var i=0;i<measures.length; i++){
10         // add the Measure to checkbox group
11         CheckboxGroup_Measures.addItem(measures[i].id,measures[i].description);
12         //add the measure to the selectedkeys
13         selectedKeys.push(measures[i].id);
14     }
15 }
16 CheckboxGroup_Measures.setSelectedKeys(selectedKeys);
17 console.log(["selectedKey ", selectedKeys]);
18 AllMeasures = selectedKeys;
19
20 // define array or the electe
21 var selectedDims = ArrayUtils.create(Type.string);
22 var dims = Table.getDataSource().getDimensions();
23 if (dims.length > 0) {
24     for (i=0;i<dims.length; i++){
25         CheckboxGroup_AllDimensions.addItem(dims[i].id,dims[i].description);
26         selectedDims.push(dims[i].id);
27     }
28 }
29
30 console.log(["selectedDims ", selectedDims]);
31 AllDimensions = selectedDims;
32
33 Utils.setMeasureFilter(selectedKeys);
34
35 Utils.setDimensionCheckboxes();

// Measures
// get all measures from the table data source
var measures = Table.getDataSource().getMeasures();

// define array or the selected Keys
var selectedKeys = ArrayUtils.create(Type.string);

if (measures.length > 0) {
    for (var i = 0; i < measures.length; i++) {
        // add the Measure to checkbox group
        CheckboxGroup_Measures.addItem(measures[i].id,
measures[i].description);
        // add the measure to the selected Keys
        selectedKeys.push(measures[i].id);
    }
}
CheckboxGroup_Measures.setSelectedKeys(selectedKeys);
console.log(["selectedKey ", selectedKeys]);
AllMeasures = selectedKeys;

// define array or the selected Keys
var selectedDims = ArrayUtils.create(Type.string);
var dims = Table.getDataSource().getDimensions();
if (dims.length > 0) {
    for (i = 0; i < dims.length; i++) {
        CheckboxGroup_AllDimensions.addItem(dims[i].id,
dims[i].description);
        selectedDims.push(dims[i].id);
    }
}

console.log(["selectedDims ", selectedDims]);
AllDimensions = selectedDims;

Utils.setMeasureFilter(selectedKeys);

Utils.setDimensionCheckboxes();
```

Now let's see how it looks like.

Click on Run Analytic Application in the upper right side of the page and the result should look something like this:

If we add the Time to the Columns Checkbox (select it in the Free Checkbox and click on add to Column), we will see that the dimension has been added and we can now see in more details what happened in which year regarding every measure.

Now, if we also add the dimension Location to the Rows, we will see the columns being filtered on the Time and the rows on the Location.

Finally, we can try to remove dimensions from the Rows and leave the Columns as we had them in the previous screenshot.

(Note: We can't remove all the dimensions from the Columns because we must filter on at least one dimension)

Now let's see how it looks like.

Click on Run Analytic Application in the upper right side of the page and the result should look something like this:

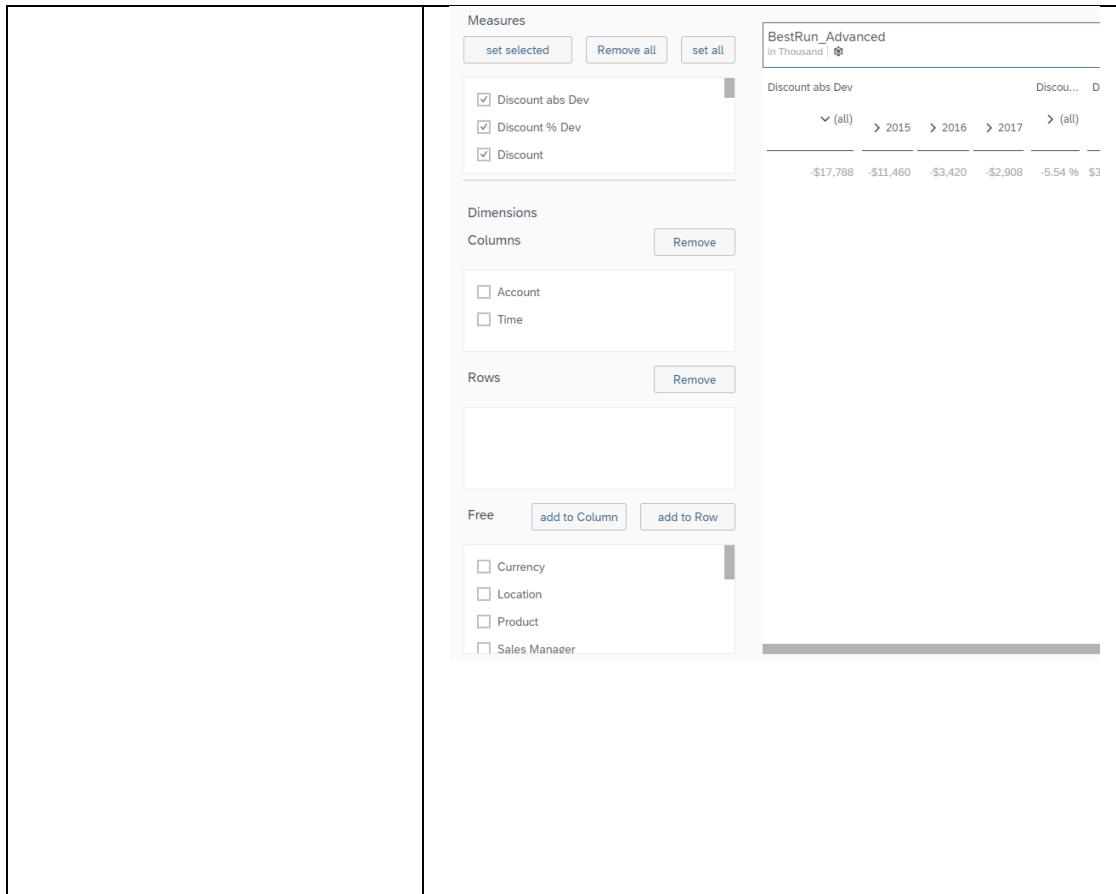
If we add the Time to the Columns Checkbox (select it in the Free Checkbox and click on add to Column), we will see that the dimension has been added and we can now see in more details what happened in which year regarding every measure.

Now, if we also add the dimension Location to the Rows, we will see the columns being filtered on the Time and the rows on the Location.

Finally, we can try to remove dimensions from the Rows and leave the Columns as we had them in the previous screenshot.

(Note: We can't remove all the dimensions from the Columns because we must filter on at least one dimension)

The screenshots show the Tableau interface with the 'Run Analytic Application' button in the top right. The first screenshot shows the initial setup with Measures (Discount abs Dev, Discount % Dev, Discount) and Dimensions (Columns: Account, Rows: Product). The second screenshot adds 'Time' to the Columns, resulting in a grid where each row is a product and each column is a quarter of the year (Q1 2015, Q2 2015, Q3 2015, Q4 2015, 2016, 2017). The third screenshot adds 'Location' to the Rows, further refining the grid by adding state-level detail for each product and time period.



6.7 Creating a Settings Panel Using a Popup Window

In this example, we will see how to use a popup window widget to create a setting panel where the user could control the contents of the Table and Chart in the Canvas.

In this use case, we want to be able to filter our table and chart according to certain measure groups of our data set. Here, *Gross Margin*, *Discount*, *Quantity Sold*, and *Original Sales Price* are the options.

These measure groups are going to be selected from a Dropdown list in our Canvas.

Afterwards, we will use the popup widget to switch between Table and Chart using a Radio Button Group and give the user the ability to control the measures (*Actual*, *Plan*, *Absolute*, and *% of Deviation*) of the measure groups using a Checkbox Group widget.

The result will look like this when we run the application:

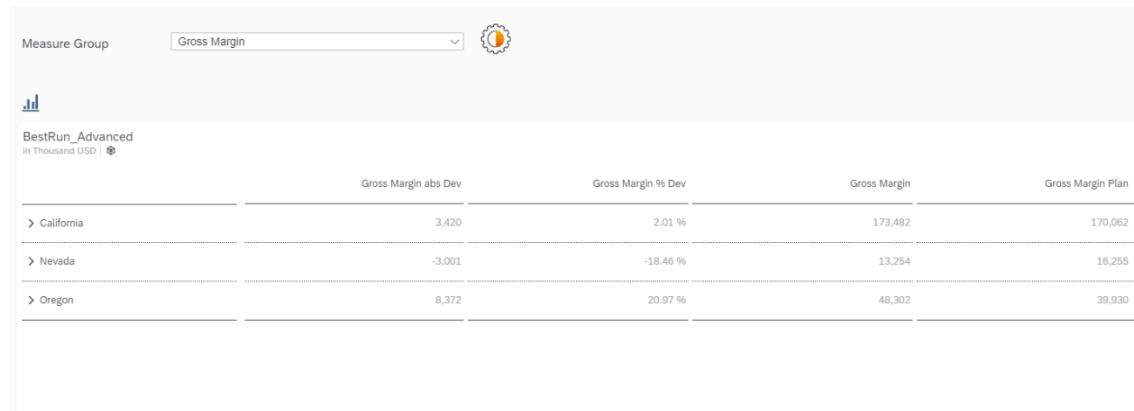


Figure 68: Example Application Settings Panel

And when the Settings button is clicked, the application will display the popup with the settings that the user can change:

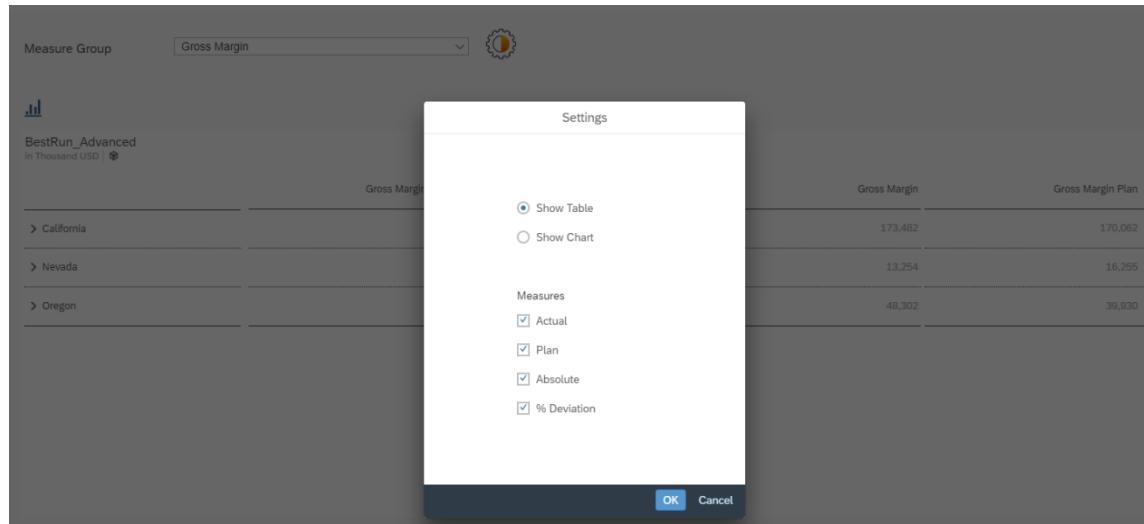
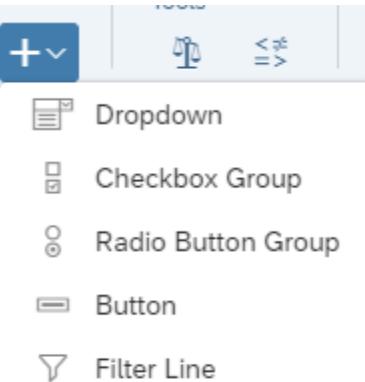
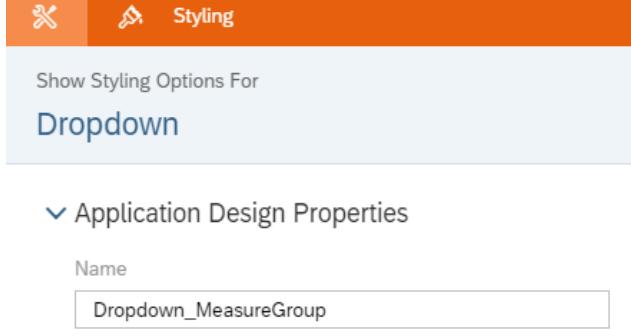


Figure 69: Popup Settings Panel

Prerequisites for this use case is having already added a table and a chart to your Canvas. To have all the functionalities in this use case, first go through the [Switching Between Chart and Table](#) exercise.

<p>The first thing we will do is add a Dropdown list that houses the measure groups with which we can filter our Table and Chart.</p> <p>To do this, click on the “+” icon in the <i>Insert</i> panel and select Dropdown and place the widget above the Table in the Canvas.</p>	 <p>The screenshot shows the 'Insert' panel with various icons. The 'Dropdown' icon, which is a blue square with a white downward arrow, is highlighted. Below the panel, a list of options is displayed:</p> <ul style="list-style-type: none">DropdownCheckbox GroupRadio Button GroupButtonFilter Line
<p>Go to the Designer (by clicking on Designer on the upper right side of the screen) and switch to the <i>Styling</i> panel by clicking on the  button.</p> <p>There, enter “Dropdown_MeasureGroup” as the Name.</p>	 <p>The screenshot shows the 'Styling' panel for a 'Dropdown' control. The top bar has tabs for 'Styling' and 'Show Styling Options For'. Below that, there is a section titled 'Application Design Properties' with a 'Name' field containing 'Dropdown_MeasureGroup'.</p>

Now, we will select which measures we want the user to be able to filter on. In this use case, we will choose 4 measures; Gross Margin, Discount, Quantity Sold, and Original Sales Price.

To enter these values in our dropdown list, switch over to the *Builder* panel by clicking on the  button.

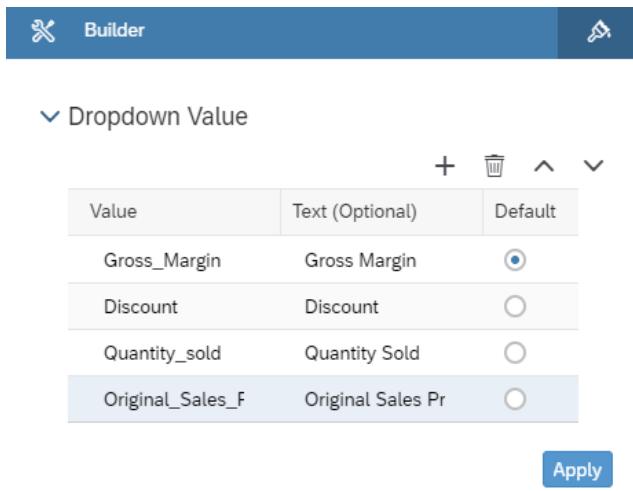
There, press the “+” icon near the Dropdown value to enter our desired values.

The first value is Gross_Margin and its displayed text should read Gross Margin.

The second value is Discount and the displayed text is the same.

The third value is Quantity_sold and its displayed text is Quantity Sold. And add a fourth Dropdown list element with the value Original_Sales_Price and its text should read Original Sales Price.

And finally, set Gross_Margin as the default value of the Dropdown list and click on Apply to save the changes.



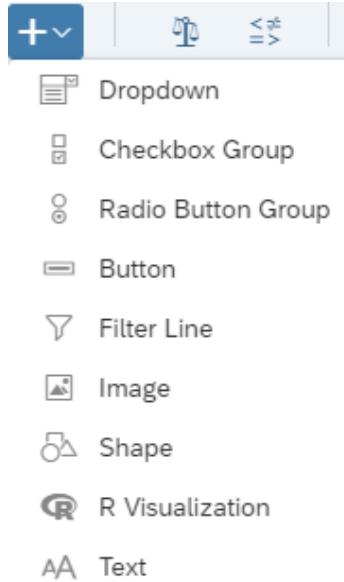
Value	Text (Optional)	Default
Gross_Margin	Gross Margin	<input checked="" type="radio"/>
Discount	Discount	<input type="radio"/>
Quantity_sold	Quantity Sold	<input type="radio"/>
Original_Sales_F	Original Sales Pr	<input type="radio"/>

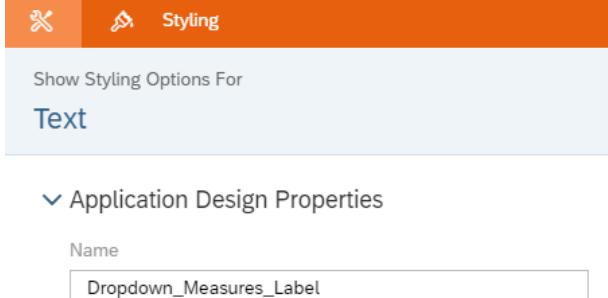
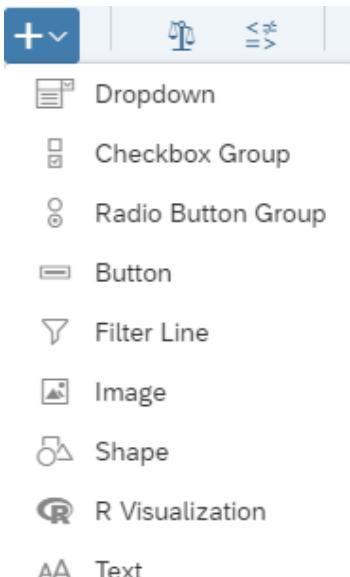
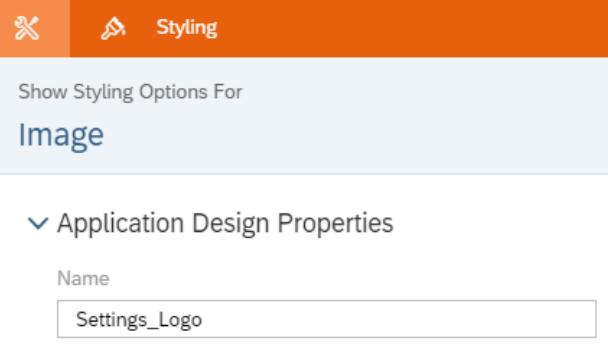
Apply

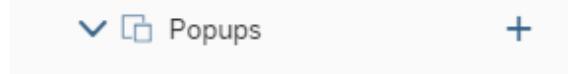
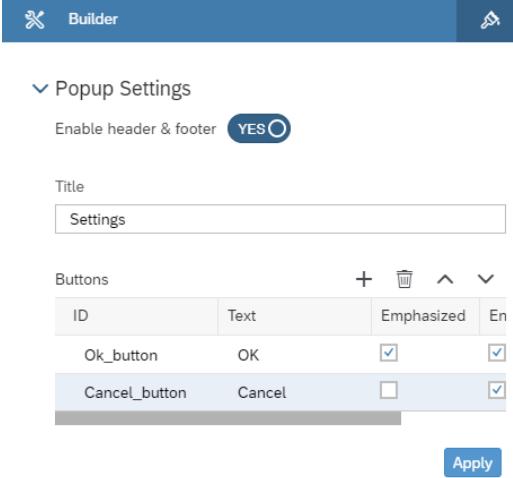
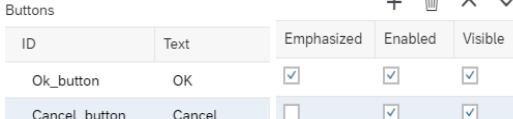
Value	Text (Optional)
Gross_Margin	Gross Margin
Discount	Discount
Quantity_sold	Quantity Sold
Original_Sales_Price	Original Sales Price

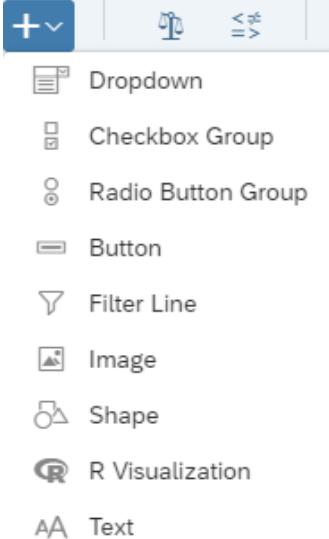
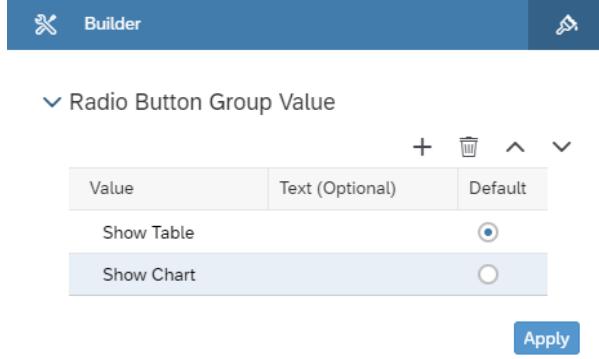
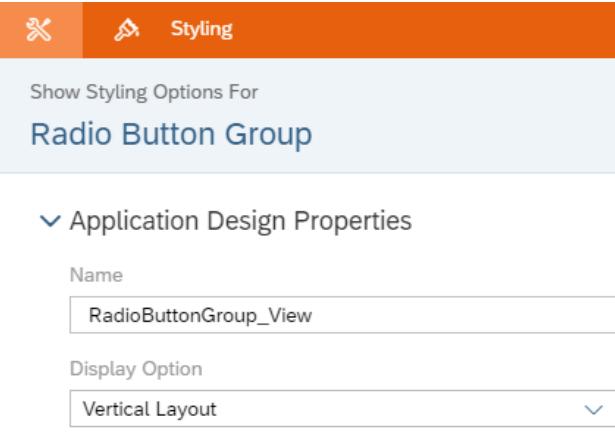
To make it clear what the contents of our Dropdown widget are, we will insert a Label.

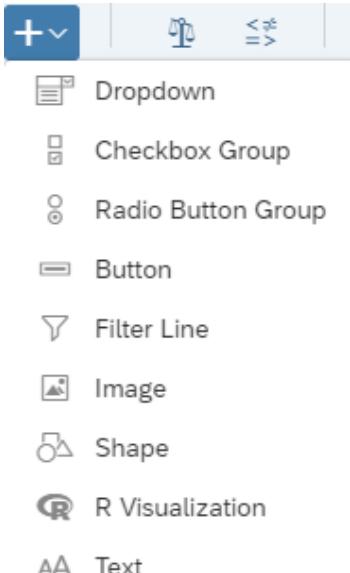
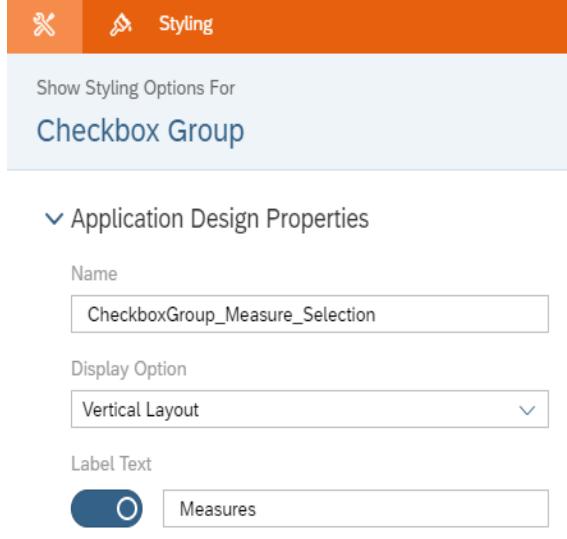
To do that, click on the “+” icon in the *Insert* panel and select Text.



<p>Place the inserted Text widget to the left side of the Dropdown widget and select it to edit its properties.</p> <p>Go to the <i>Styling</i> panel and enter “Dropdown_Measures_Label” as the Name.</p>	
<p>To edit what the label shows, double click on the Text widget in the Canvas and enter “Measure Group”.</p>	
<p>The next step is adding the popup that lets us edit some settings of our Table and Chart.</p> <p>However, we first need to add an icon that, when the user clicks on, will make the popup appear.</p> <p>To do this, click on the “+” icon in the <i>Insert</i> panel and choose <i>Image</i>.</p> <p>Now add any image that you want to use as an icon for the settings.</p> <p>In our application, we used  as our image.</p>	
<p>To edit the name of the image, go to the <i>Styling</i> panel and enter Settings_Logo as the name.</p>	

<p>Now we will add our popup window. To do this, look for Popups in the Layout and click on the "+" icon to add one.</p>	
<p>Double click on the newly added popup in the Layout to rename it and enter the name "Popup_Settings".</p>	
<p>And then select the popup in the Layout and a new window should open. There, we will add the elements that we want to appear in our Popup window.</p> <p>To have a header and a footer, click on the popup and go to the <i>Builder</i> panel. There, toggle the "Enable header & footer" button to YES.</p>	
<p>Enter "Settings" as the Title of the Popup.</p>	
<p>We will have two buttons, an Ok and a Cancel button. To add them, click on the "+" icon next to Buttons. Set the ID of the first button to "Ok_button" and the text to "OK". Finally, select all the options displayed afterwards (Emphasized, Enabled, and Visible).</p>	
<p>Set the ID of the second button to "Cancel_button" and the text to "Cancel". Select the options Enabled and Visible but leave the Emphasized checkbox disabled. Click on Apply to save the changes.</p>	

<p>In this settings popup, we would like to give the user the option of switching between the Chart and Table. To achieve this, add a Radio Button Group widget from the <i>Insert</i> panel and place it in the middle of the Popup window.</p>	
<p>To edit the properties of the Radio Button Group, select the widget and go to the <i>Builder</i> panel. There, we will add the two options that users can choose from. The first will be "Show Table" and we'll set this to the default while the second will read "Show Chart". After entering the values, click on Apply to save the changes.</p>	
<p>To edit the properties of the Radio Button Group, switch over to the <i>Styling</i> panel. There, enter "RadioButtonGroup_View" as the Name and select "Vertical Layout" as the Display Option.</p>	

<p>We will also give the user the option of choosing which measures of the chosen measure group they want displayed.</p> <p>To do this, we will add a Checkbox Group from the <i>Insert</i> panel and place it underneath the Radio Button Group widget.</p>	
<p>We will firstly edit the <i>Styling</i> panel properties of the widget. To do that, select it in the Canvas or the Layout and go to the <i>Styling</i> panel.</p> <p>There, enter “CheckboxGroup_Measure_Selection” as the Name and select “Vertical Layout” as the Display Option.</p> <p>We also want a label text, so we will toggle the Label text option to enable it and write “Measures” to display it as our Checkbox Group label.</p>	

The next step is to edit the values that appear in the Checkbox. To do this, go to the *Builder* panel. We will add the values Actual, Plan, Absolute, and % Deviation as the measures with which the measure groups can be filtered. To do that, click on the "+" button and add the values. We will set all 4 values as Default. Click on Apply to save the changes.

Value	Text (Optional)
Actual	Actual
Plan	Plan
_Abs	Absolute
_Percent	% Deviation

Apply

Value	Text (Optional)
Actual	Actual
Plan	Plan
_Abs	Absolute
_Percent	% Deviation

We also need to write a script for the settings icon we added so that when the user clicks on it, the settings popup we added, is opened.

To do that, select the icon in the Layout and click on the icon next to it.

There, we will simply make the click event open our settings popup.

```

    <script>
        function onClick() : void {
            Popup_Settings.open();
        }
    </script>

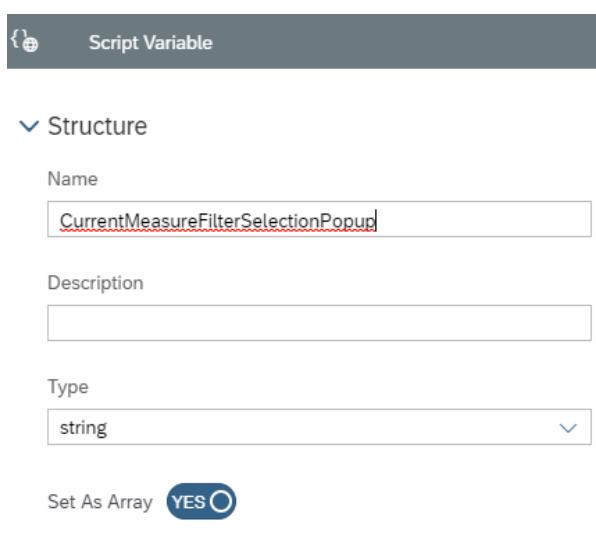
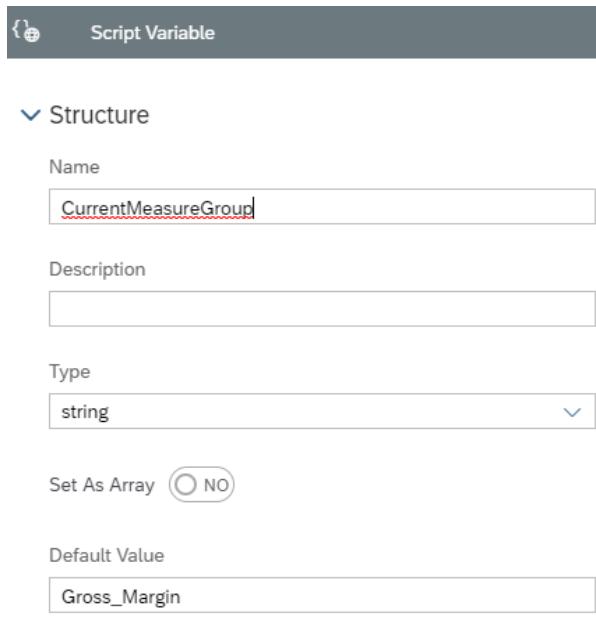
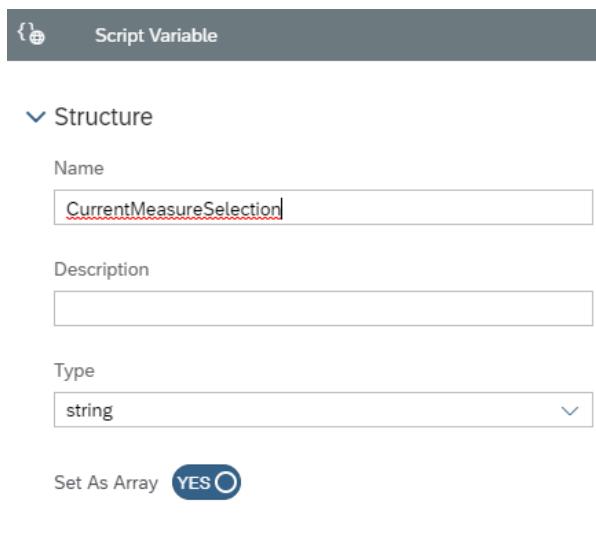
```

To be able to access all the selections that the user made from any widget in our app, we need to add global variables.

To add these script variables, go to Scripting and click the "+" next to Script Variables.

Script Variables

+

<p>The first script variable we will add, is one that will hold the concatenated filter of the Dropdown List in the Canvas and the Checkbox Group in the Popup window.</p> <p>Add a script variable and its properties enter "CurrentMeasureFilterSelectionPopup" in the Name field, set the Type to "string", and toggle the Set As Array button to "YES".</p>	 <p>Script Variable</p> <p>Structure</p> <p>Name: CurrentMeasureFilterSelectionPopup</p> <p>Description:</p> <p>Type: string</p> <p>Set As Array: YES</p>
<p>The second script variable we will add is one that will hold the current measure filter from the Dropdown list.</p> <p>Add a script variable and in its properties enter "CurrentMeasureGroup" in the Name field, set the Type to "string", and the Default Value to "Gross_Margin".</p>	 <p>Script Variable</p> <p>Structure</p> <p>Name: CurrentMeasureGroup</p> <p>Description:</p> <p>Type: string</p> <p>Set As Array: NO</p> <p>Default Value: Gross_Margin</p>
<p>The third and final script variable we need is one that will hold the measures selected from the Checkbox Group in the Popup window.</p> <p>Add a script variable and its properties enter "CurrentMeasureSelection" in the Name field, set the Type to "string", and toggle the Set As Array button to "YES".</p>	 <p>Script Variable</p> <p>Structure</p> <p>Name: CurrentMeasureSelection</p> <p>Description:</p> <p>Type: string</p> <p>Set As Array: YES</p>

To define what should happen when a filter is selected, we need to create a Script Object.

In this object, we will create a function that sets the measure filter according to what the user has chosen from the Checkbox Group.

To create a Script Object, select the “+” icon next to SCRIPT OBJECTS under the Layout.

Afterwards, rename both the folder that was created as well as the function.

We will name the folder Utils and the function setMeasureFilter.

To rename the objects, hover over them one by one and when the  icon appears click on it and choose *Rename*.



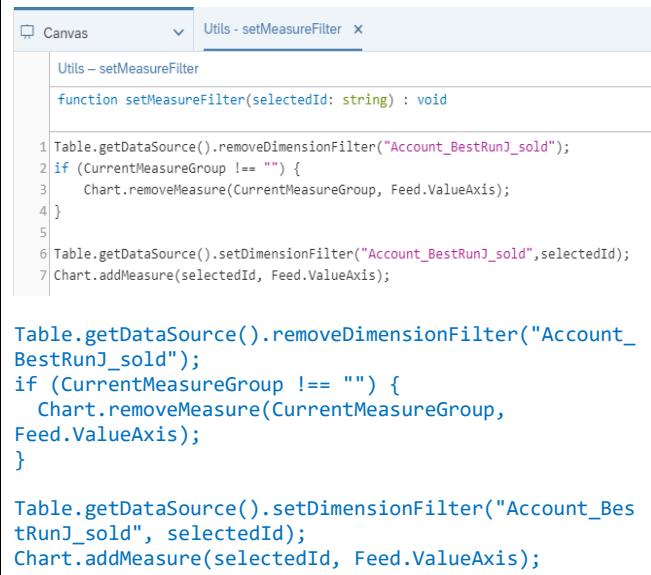
Click on the function setMeasureFilter and when the Editing window opens, click on the "+" icon next to Arguments.

Here, we will add an argument with the name "selectedId" and the Type string.

The screenshot shows a software interface for managing script functions. On the left, there's a vertical toolbar with icons for Home, Recent, Create, and Help. The main area has a dark header bar with the title '(X) Script Function'. Below the header, there are two tabs: 'Script Object' and 'Utils', with 'Utils' selected. A large 'Properties' section is open, containing fields for Name (setMeasureFilter), Description (empty), Return Type (void), and a 'Set As Array' toggle switch (set to NO). A '+' button is located at the bottom right of the Properties section. Below this is the 'Arguments' section, which is currently collapsed. It contains a sub-header 'Argument' with a back arrow icon, and a list entry for 'Utils – setMeasureFilter'. Underneath the Arguments section is another collapsed section labeled 'Settings'. The entire interface has a clean, modern design with a light gray background and white text.

To define what the setMeasureFilter function does, go to the function in the Layout, hover over its name, and click on the  icon next to it.

In this use case, when the setMeasureFilter function is called, the set measure filters are removed from the Table and the Chart and the selected measure sent to the function is inserted instead.



The screenshot shows a code editor window with two tabs: "Canvas" and "Utils - setMeasureFilter". The "Utils - setMeasureFilter" tab is active. The code is written in TypeScript and defines a function that removes a specific dimension filter from both a Table and a Chart, and then adds a new one.

```
Utils - setMeasureFilter
function setMeasureFilter(selectedId: string) : void
1 Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold");
2 if (CurrentMeasureGroup !== "") {
3     Chart.removeMeasure(CurrentMeasureGroup, Feed.ValueAxis);
4 }
5
6 Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold",selectedId);
7 Chart.addMeasure(selectedId, Feed.ValueAxis);

Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold");
if (CurrentMeasureGroup !== "") {
    Chart.removeMeasure(CurrentMeasureGroup,
Feed.ValueAxis);
}

Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold", selectedId);
Chart.addMeasure(selectedId, Feed.ValueAxis);
```

Now, we will define what happens when a user selects a measure group from the Dropdown list in the Canvas. To do that, select the Dropdown widget in the Canvas or Layout and click on the

 icon that appears next to it.

In this script, we will first see which value was selected and will remove the measures of these measure groups from our Chart.

Then, we will save the current selection in our script variable, CurrentMeasureGroup.

Afterwards, we will see which measures were selected in the Checkbox in the Popup so that we filter on all the inputs the user gave us.

After getting these values, we will remove any old filters used and apply the new ones.

To get a valid filter, we will concatenate the selected measures to a filter statement.

Finally, we will save the concatenated filter statement in our CurrentMeasureFilterSelectionPopup script variable and the selected keys of the Checkbox Group in the CurrentMeasureSelection script variable.

```
Canvas Dropdown_MeasureGroup - o...
Dropdown_MeasureGroup – onSelect
Called by the system when the user selects an entry in the dropdown.
function onSelect(): void

1 var sel = Dropdown_MeasureGroup.getSelectedKey();
2
3 if (CurrentMeasureGroup === 'Gross_Margin') {
4     Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginActual]", Feed.ValueAxis);
5     Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginPlan]", Feed.ValueAxis);
6     Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginAbs]", Feed.ValueAxis);
7     Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginPercent]", Feed.ValueAxis);
8 }
9 else if (CurrentMeasureGroup === 'Discount') {
10    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountActual]", Feed.ValueAxis);
11    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountPlan]", Feed.ValueAxis);
12    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountAbs]", Feed.ValueAxis);
13    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountPercent]", Feed.ValueAxis);
14 }
15 else if (CurrentMeasureGroup === 'Original_Sales_Price') {
16    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[OriginalSalesPriceActual]", Feed.ValueAxis);
17    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[OriginalSalesPricePlan]", Feed.ValueAxis);
18    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[OriginalSalesPriceAbs]", Feed.ValueAxis);
19    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[OriginalSalesPricePercent]", Feed.ValueAxis);
20 }
21 else if (CurrentMeasureGroup === 'Quantity_Sold') {
22    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[QuantitySoldActual]", Feed.ValueAxis);
23    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[QuantitySoldPlan]", Feed.ValueAxis);
24    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[QuantitySoldAbs]", Feed.ValueAxis);
25    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[QuantitySoldPercent]", Feed.ValueAxis);
26 }
27
28 // save the current selection (measure filter) into a global variable
30 CurrentMeasureGroup = sel;
31
32 // get Measures Selection
33 var Selected_Measures = CheckboxGroup_Measure_Selection.getSelectedKeys();
34
35 // remove the current measures from chart
36 for (var i=0; i<CurrentMeasureFilterSelectionPopup.length; i++) {
37     Chart.removeMeasure(CurrentMeasureFilterSelectionPopup[i], Feed.ValueAxis);
38 }
39
40 // help variables
41 var Filter_Pattern_1 = "[Account_BestRunJ_sold].[parentId].[";
42 var Filter_Pattern_2 = "]";
43 var Filter_Area = ArrayUtil.create(Type.String);
44
45 // loop over the selected measures
46 for (i=0; i<Selected_Measures.length; i++) {
47     //concat all selection information together to a valid filter statement
48     var Filter = Filter_Pattern_1 + CurrentMeasureGroup + Selected_Measures[i] + Filter_Pattern_2;
49     Filter_Area.push(Filter);
50 }
51
52 // add Measure to Chart
53 Chart.addMeasure(Filter, Feed.ValueAxis);
54
55 // remove the "old" filter and set the new filter selection
56 Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold");
57 Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold", Filter_Area);
58
59 // save the current measure filter selection into a global variable
60 // Note --> this global variable need to be set with the default values on the onInitialization event from the Main Canvas
61 CurrentMeasureFilterSelectionPopup = Filter_Area;
62 CurrentMeasureSelection = Selected_Measures;
63
64 // write the current measure filter selection to the browser console
65 console.log("Measure Selection: ", CurrentMeasureSelection);
66 console.log("Measure Filter Selection: ", CurrentMeasureFilterSelectionPopup);

var sel = Dropdown_MeasureGroup.getSelectedKey();

if (CurrentMeasureGroup === 'Gross_Margin') {

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginActual]", Feed.ValueAxis);

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginPlan]", Feed.ValueAxis);

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginAbs]", Feed.ValueAxis);

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[GrossMarginPercent]", Feed.ValueAxis);
}

else if (CurrentMeasureGroup === 'Discount') {

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountActual]", Feed.ValueAxis);

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountPlan]", Feed.ValueAxis);

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountAbs]", Feed.ValueAxis);

    Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[DiscountPercent]", Feed.ValueAxis);
}

else if (CurrentMeasureGroup === 'Quantity_Sold') {
```

```

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Quantity_soldActual]", Feed.ValueAxis);

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Quantity_soldPlan]", Feed.ValueAxis);

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Quantity_sold_Abs]", Feed.ValueAxis);

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Quantity_sold_Percent]", Feed.ValueAxis);
}
else if (CurrentMeasureGroup ===
'Original_Sales_Price') {

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Original_Sales_PriceActual]", Feed.ValueAxis);

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Original_Sales_PricePlan]", Feed.ValueAxis);

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Original_Sales_Price_Abs]", Feed.ValueAxis);

Chart.removeMeasure("[Account_BestRunJ_sold].[parentId].[Original_Sales_Price_Percent]", Feed.ValueAxis);
}

// save the current selection (measure filter) into a
global variable
CurrentMeasureGroup = sel;

// get Measures Selection
var Selected_Measures =
CheckboxGroup_Measure_Selection.getSelectedKeys();

// remove the current measures from Chart
for (var i = 0; i <
CurrentMeasureFilterSelectionPopup.length; i++) {

Chart.removeMeasure(CurrentMeasureFilterSelectionPopu
p[i], Feed.ValueAxis);
}

// help variables
var Filter_Pattern_1 =
"[Account_BestRunJ_sold].[parentId].&";
var Filter_Pattern_2 = "]";
var Filter_Area = ArrayUtils.create(Type.string);

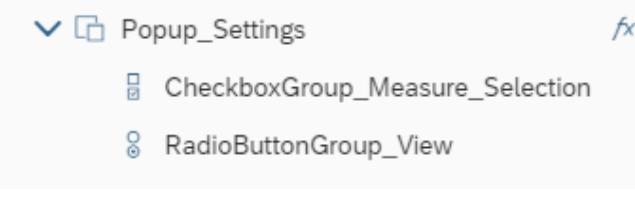
// loop over the selected measures
for (i = 0; i < Selected_Measures.length; i++) {
    // concate all selection information together to a
    valid filter statemant
    var Filter = Filter_Pattern_1 + CurrentMeasureGroup
+ Selected_Measures[i] + Filter_Pattern_2;
    Filter_Area.push(Filter);

    // add Measure to Chart
    Chart.addMeasure(Filter, Feed.ValueAxis);
}

// remove the "old" filter and set the new filter
selection
Table.getDataSource().removeDimensionFilter("Account_
BestRunJ_sold");
Table.getDataSource().setDimensionFilter("Account_Bes
tRunJ_sold", Filter_Area);

// save the current measure filter selection into a
global variable
// Note -> this global variable needs to be set with
the default values on the onInitialization event from
the Main Canvas
CurrentMeasureFilterSelectionPopup = Filter_Area;

```

	<pre>CurrentMeasureSelection = Selected_Measures; // write the current measure filter selection to the // browser console console.log(["Measure Selection: ", CurrentMeasureSelection]); console.log(["Measure Filter Selection: ", CurrentMeasureFilterSelectionPopup]);</pre>
The final script we need to write is the script of buttons OK and Cancel that we have in our popup window. Select the popup in the Layout and click on the  icon that appears next to it.	

We have two buttons, OK and Cancel, so, we will start off with an if statement that differentiates the buttons according to their IDs.

In this script, we will get the selections from the Checkbox Group in the popup window and then we will remove the measures currently being used as filters for the Chart.

To get a valid filter, we will concatenate the selected measures to a filter statement.

We will save the concatenated filter statement in our CurrentMeasureFilterSelectionPopup script variable and the selected keys of the Checkbox Group in the CurrentMeasureSelection script variable.

Afterwards, we will get the selected key of the Radio Button Group in the Popup window. If "Show Table" is selected, then we will set the Table to visible and the Chart to invisible and vice versa if "Show Chart" is selected.

Finally, we will close the Popup whether the user clicked on OK or Cancel.

```
↳ Popup_Settings -> Popup_Settings - onButtonClick x
Popup_Settings - onButtonClick
Called when the user clicks one of the buttons in the footer of the popup.
Function onButtonClick(buttonId: string) : void

1 if (buttonId === "OK_button"){
2
3 // get Measures Selection
4 var Selected_Measures = CheckboxGroup_Measure_Selection.getSelectedKeys();
5
6 if (CurrentMeasureSelection !== Selected_Measures) {
7
8 // remove the current measures from Chart
9 for (var i=0; i<CurrentMeasureGroup.length; i++){
10 Chart.removeMeasure(CurrentMeasureFilterSelectionPopup[i], Feed.ValueAxis);
11 }
12
13 // help variables
14 var Filter_Pattern_1 = "[Account_BestRunJ_sold].[parentId].&";
15 var Filter_Pattern_2 = "]";
16 var Filter_Area = ArrayUtils.create(Type.string);
17
18 // loop over the selected measures
19 for (i=0; i<Selected_Measures.length; i++){
20
21 //concat all selection information together to a valid filter statement
22 var Filter = Filter_Pattern_1 + CurrentMeasureGroup + Selected_Measures[i] + Filter_Pattern_2;
23 Filter_Area.push(Filter);
24 // add Measure to Chart
25 Chart.addMeasure(Filter, Feed.ValueAxis);
26 }
27
28 // remove the "old" filter and set the new filter selection
29 Table.getDataSource().removeDimensionFilter("Account_BestRunJ_sold");
30 Table.getDataSource().setDimensionFilter("Account_BestRunJ_sold", Filter_Area);
31
32 // save the current measure filter selection into a global variable
33 // Note --> this global variable need to be set with the default values on the onInitialization event from the Main Canvas
34 CurrentMeasureFilterSelectionPopup = Filter_Area;
35 CurrentMeasureSelection = Selected_Measures;
36
37 // write the current measure filter selection to the browser console
38 console.log(["Measure Selection: ", CurrentMeasureSelection]);
39 console.log(["Measure Filter Selection: ", CurrentMeasureFilterSelectionPopup]);
40
41 }
42
43 // set the visibility of Chart and Table --> Script from the RadioButtonGroup_View onSelect event
44 var sel = RadioButtonGroup_View.getSelectedKey();
45
46 if (sel === "Show Table") {
47 Table.setVisible(true);
48 Chart.setVisible(false);
49 }
50 else {
51 Table.setVisible(false);
52 Chart.setVisible(true);
53 }
54
55
56 Popup_Settings.close();}

if (buttonId === "Ok_button") {
// get Measures Selection
var Selected_Measures =
CheckboxGroup_Measure_Selection.getSelectedKeys();

if (CurrentMeasureSelection !== Selected_Measures)

{
// remove the current measures from Chart
for (var i = 0; i < CurrentMeasureGroup.length;
i++) {

Chart.removeMeasure(CurrentMeasureFilterSelectionPopu
p[i], Feed.ValueAxis);

// help variables
var Filter_Pattern_1 =
"[Account_BestRunJ_sold].[parentId].&";
var Filter_Pattern_2 = "]";
var Filter_Area = ArrayUtils.create(Type.string);

// loop over the selected measures
for (i = 0; i < Selected_Measures.length; i++) {
// concat all selection information together
to a valid filter statement
var Filter = Filter_Pattern_1 +
CurrentMeasureGroup + Selected_Measures[i] +
Filter_Pattern_2;
Filter_Area.push(Filter);
// add Measure to Chart
Chart.addMeasure(Filter, Feed.ValueAxis);
}
}
```

```
// remove the "old" filter and set the new filter  
selection  
  
Table.getDataSource().removeDimensionFilter("Account_  
BestRunJ_sold");  
  
Table.getDataSource().setDimensionFilter("Account_Bes  
tRunJ_sold", Filter_Area);  
  
    // save the current measure filter selection into  
a global variable  
    // Note -> this global variable needs to be set  
with the default values on the onInitialization event  
from the Main Canvas  
    CurrentMeasureFilterSelectionPopup = Filter_Area;  
    CurrentMeasureSelection = Selected_Measures;  
  
    // write the current measure filter selection to  
the browser console  
    console.log(["Measure Selection: ",  
CurrentMeasureSelection]);  
    console.log(["Measure Filter Selection: ",  
CurrentMeasureFilterSelectionPopup]);  
}  
  
// set the visibility of Chart and Table -> Script  
from the RadioButtonGroup_View onSelect event  
var sel = RadioButtonGroup_View.getSelectedKey();  
  
if (sel === 'Show Table') {  
    Table.setVisible(true);  
    Chart.setVisible(false);  
}  
else {  
    Table.setVisible(false);  
    Chart.setVisible(true);  
}  
}  
Popup_Settings.close();
```

Now let's see how it looks like.

Click on Run Analytic Application in the upper right side of the page and the result should look something like this:

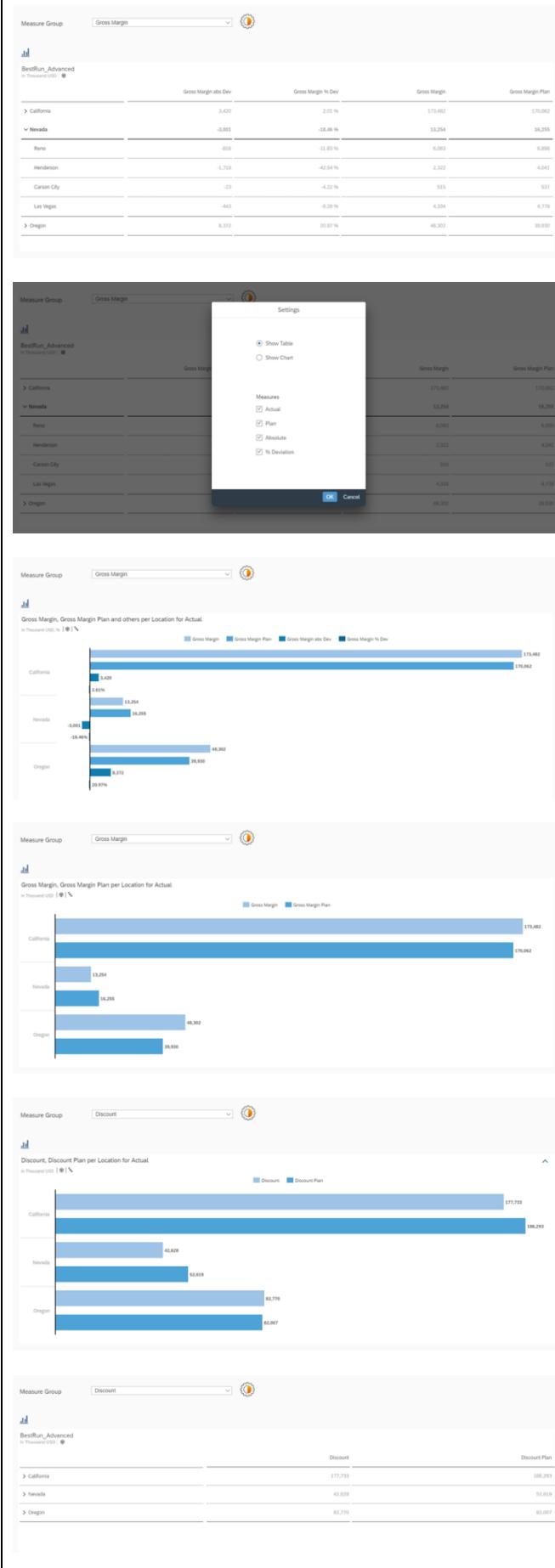
If we click on the Settings icon, the Popup will appear.

Now, let's select "Show Chart" from the popup window and leave all the measures selected. The result should be that the settings are left as they were, and the only change is that the Chart is now displayed.

Open the popup window again but this time select only two items from the Checkbox Group.
Here, we have selected Actual and Plan.

Now, change the Measure Group from the "Gross Margin" to "Discount" and the two measures, Actual and Plan are displayed here for the measure group Discount.

Finally, let's switch back to the Table from the popup window while leaving all the settings unchanged from the previous example.
The result is that the Discount measure group is presented and only Actual and Plan are displayed.



6.8 Selection Handling in a Table or Chart and Open a Details Popup

In this example, we will let the user select certain elements in the Table and the Chart that when clicked on, open a popup window with extra information in a chart format about the selected element.

In a Table, a user will be able to select a measure cell, a dimension cell, or a data cell. Each will open a popup window that displays information about the selected element in a trend chart.

In the Chart, a user will be able to select a dimension cell and a measure/dimension chart bar (for example, *Gross Margin Plan for Lemonade*).

There are also two Dropdown lists, one for dimensions and the other for hierarchies. The list of dimensions let the user choose which dimension filter they want to use on the Table/Chart. In this use case, we have chosen 4 dimensions; *Location*, *Product*, *Store*, and *Sales Manager*.

The second Dropdown list displays the available hierarchies that can be used to change how the data is displayed.

Note: In this example, only single selection is supported for the Table and Chart.

The result will look like this when we run the application:

Location	Gross Margin Plan	Gross Margin	Gross Margin abs Dev	Gross Margin % Dev
Los Angeles	48,542	48,972	430	0.89 %
Reno	6,898	6,083	-816	-11.83 %
Henderson	4,041	2,322	-1,719	-42.54 %
Carson City	537	515	-22	-4.22 %
Portland	9,097	10,499	1,402	15.41 %
Salem	14,680	19,488	4,807	32.75 %
Eugene	7,539	8,924	1,385	18.38 %
Gresham	2,279	4,483	2,204	96.75 %
Hillsboro	537	820	283	52.56 %
Beaverton	5,797	4,088	-1,710	-29.49 %
San Francisco	21,391	19,620	-1,771	-8.28 %
San Diego	14,872	17,802	2,930	19.70 %
Sacramento	21,484	24,859	3,374	15.71 %
San Jose	27,659	24,802	-2,857	-10.33 %
Oakland	12,677	12,754	78	0.61 %
Santa Barbara	9,676	11,174	1,498	15.48 %
Beverly Hills	13,760	13,498	-262	-1.90 %
Las Vegas	4,778	4,334	-443	-9.28 %

Figure 70: Example Application Details Popup

And when a cell is chosen, a popup window like the one in the screenshot will appear (In this screenshot, the dimension cell of *Los Angeles* was clicked on in the Table):

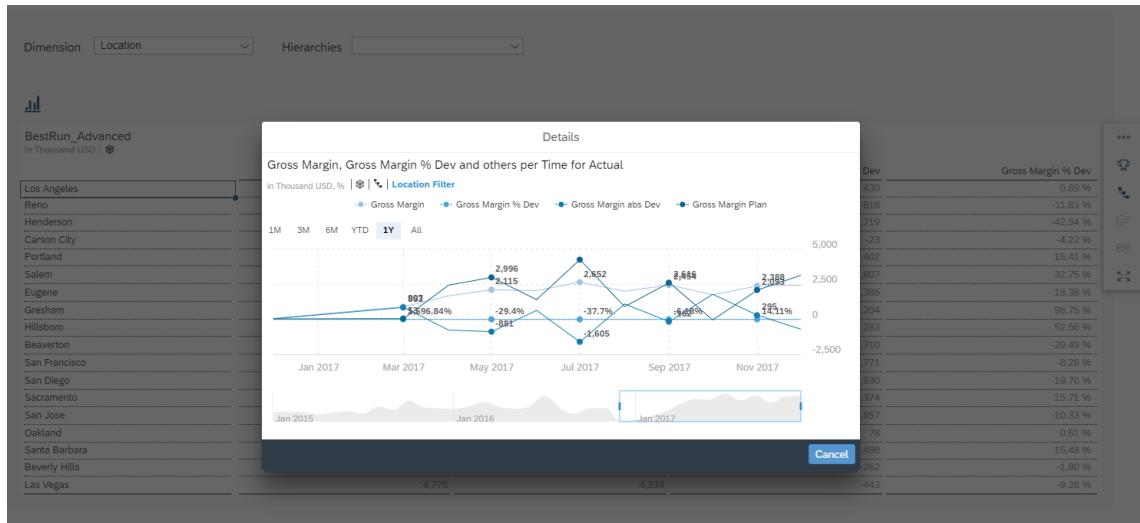
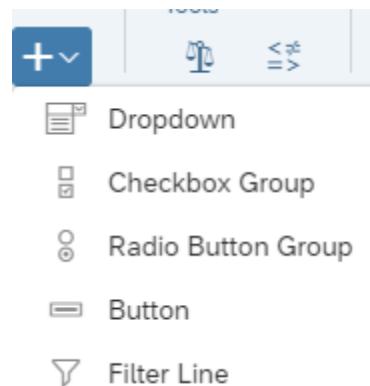


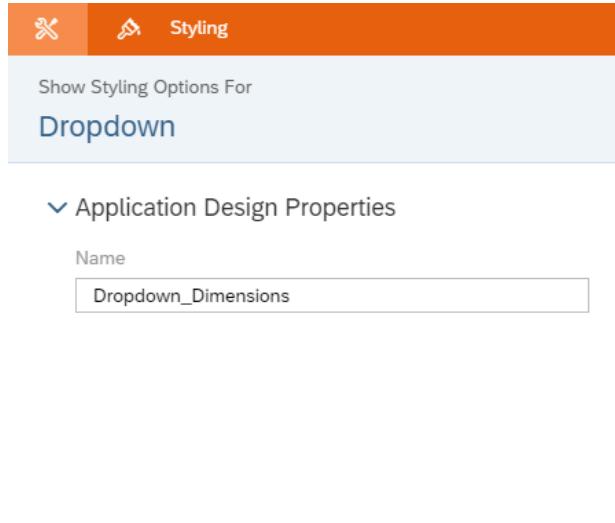
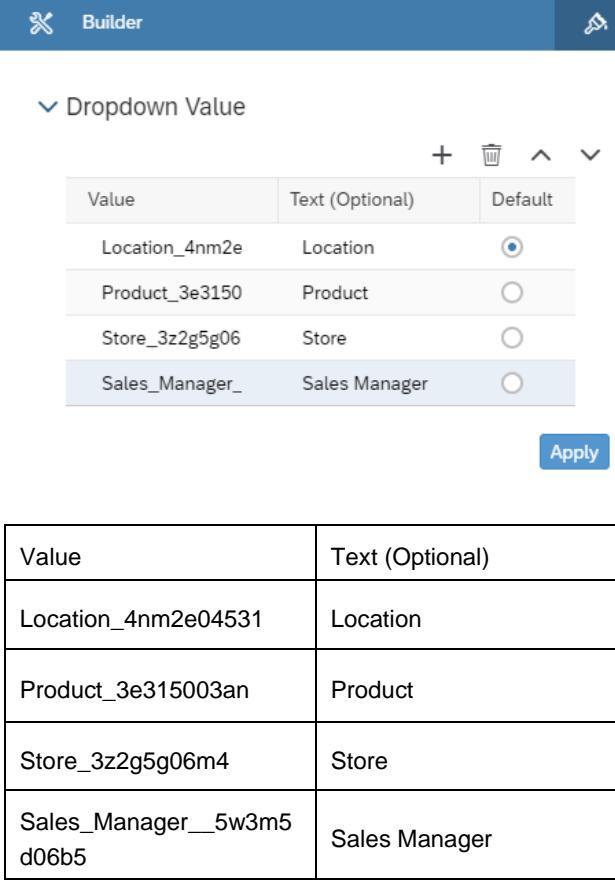
Figure 71: Details Popup

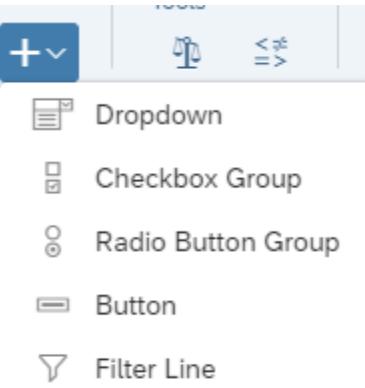
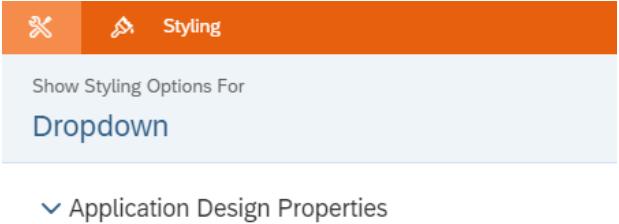
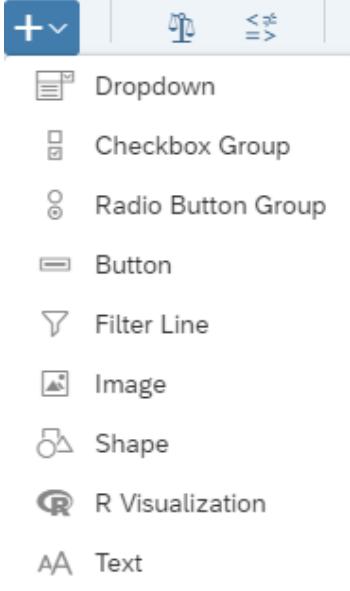
Prerequisites for this use case is having already added a functioning Chart and Table to your Canvas. To have all the functionalities in this use case, first go through the [Switching Between Chart and Table](#) exercise.

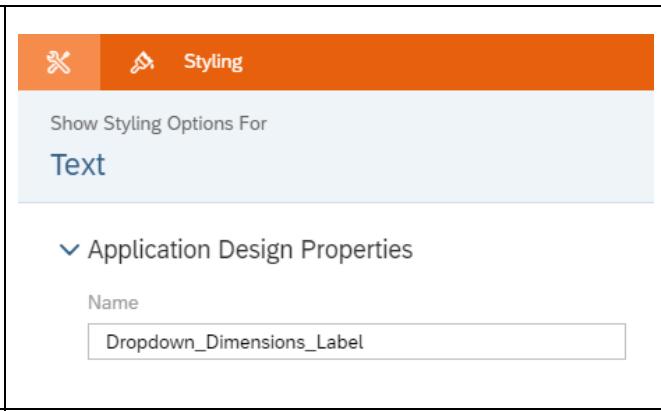
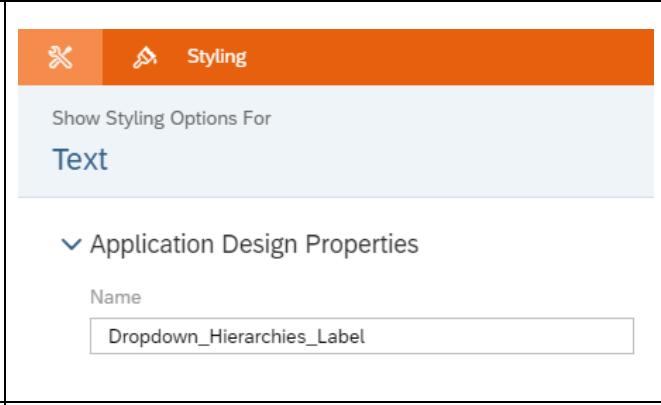
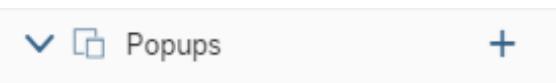
We recommend using the same names as that exercise for the Chart and Table Chart so that the scripts in this use case don't have to be altered.

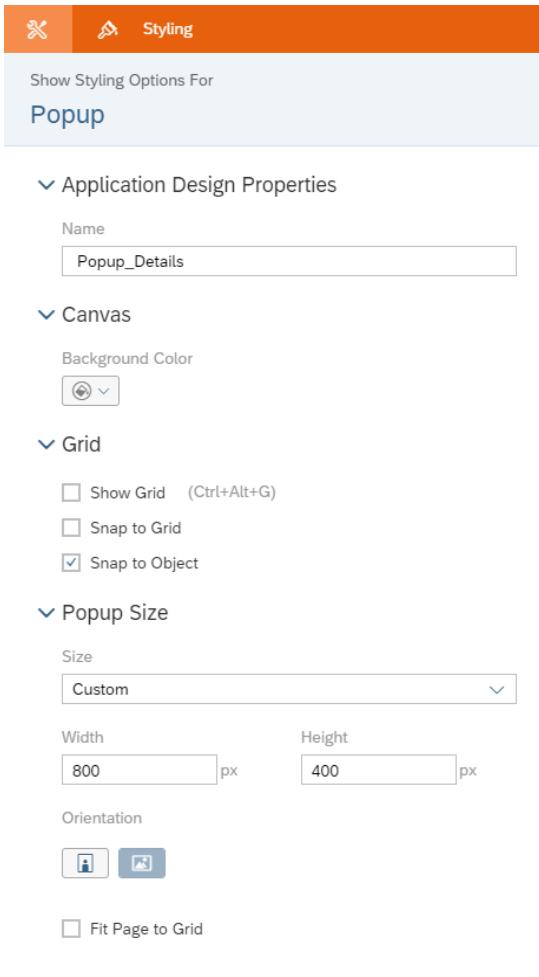
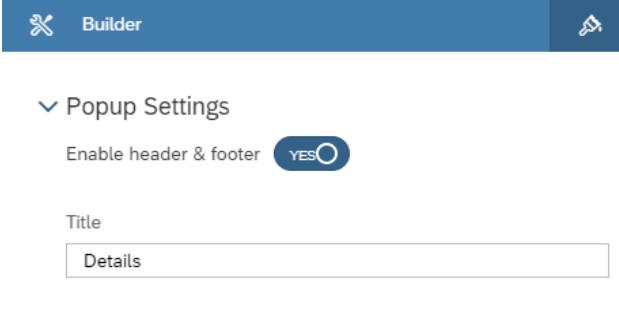
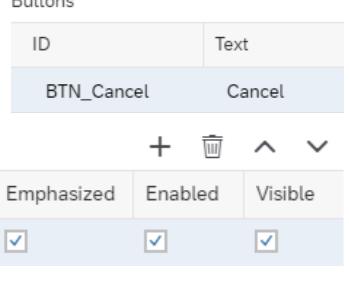
The first thing we will do is add a Dropdown list that houses the dimensions in our data set. To do this, click on the "+" icon in the *Insert* panel and select Dropdown and place the widget above the Table in the Canvas.

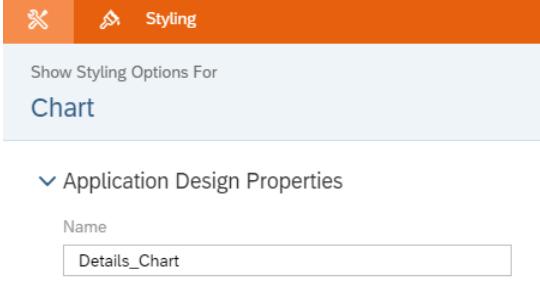
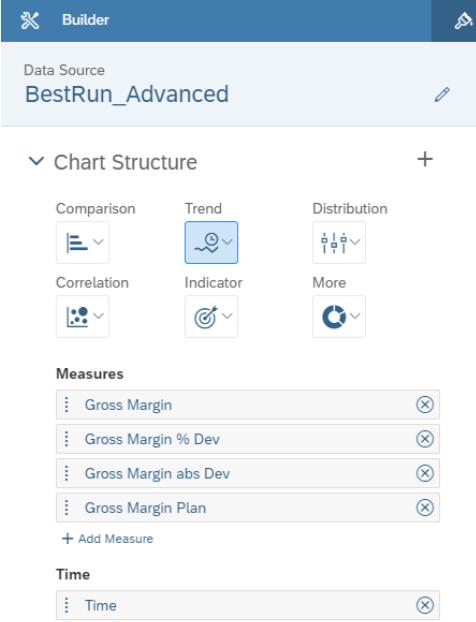


<p>Go to the Designer (by clicking on Designer on the upper right side of the screen) and switch to the <i>Styling</i> panel by clicking on the  button.</p> <p>There, enter “Dropdown_Dimensions” as the Name.</p>											
<p>Now, we will select which dimensions we want the user to be able to filter on. In this use case, we will choose 4; Location, Product, Store, and Sales Manager.</p> <p>To enter these values in our dropdown list, switch over to the <i>Builder</i> panel by clicking on the  button.</p> <p>There, press the “+” icon near the Dropdown value to enter the desired values.</p> <p>The first value is Location_4nm2e04531 and its displayed text should read Location.</p> <p>The second value is Product_3e315003an and the displayed text is Product.</p> <p>The third value is Store_3z2g5g06m4 and its displayed text is Store.</p> <p>And we'll add a fourth Dropdown list element with the value Sales_Manager_5w3m5d06b5 and its text should read Sales Manager.</p> <p>And finally, set Location as the default value of the Dropdown list.</p> <p>Click on Apply to save the changes.</p>	 <table border="1" data-bbox="752 1298 1367 1653"> <thead> <tr> <th>Value</th> <th>Text (Optional)</th> </tr> </thead> <tbody> <tr> <td>Location_4nm2e04531</td> <td>Location</td> </tr> <tr> <td>Product_3e315003an</td> <td>Product</td> </tr> <tr> <td>Store_3z2g5g06m4</td> <td>Store</td> </tr> <tr> <td>Sales_Manager_5w3m5d06b5</td> <td>Sales Manager</td> </tr> </tbody> </table>	Value	Text (Optional)	Location_4nm2e04531	Location	Product_3e315003an	Product	Store_3z2g5g06m4	Store	Sales_Manager_5w3m5d06b5	Sales Manager
Value	Text (Optional)										
Location_4nm2e04531	Location										
Product_3e315003an	Product										
Store_3z2g5g06m4	Store										
Sales_Manager_5w3m5d06b5	Sales Manager										

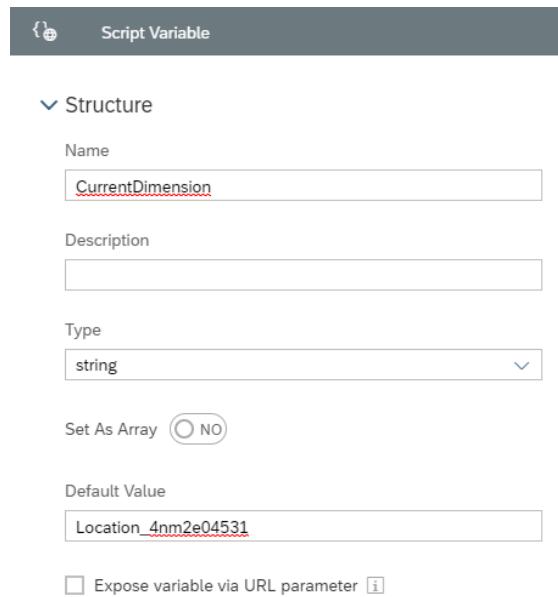
<p>We will now add a second list where the user can choose the hierarchy in which they want to display their data.</p> <p>To do this, click on the "+" icon in the <i>Insert</i> panel and select Dropdown and place the widget next to the first Dropdown list in the Canvas.</p>	
<p>Go to the Designer (by clicking on Designer on the upper right side of the screen) and switch to the <i>Styling</i> panel by clicking on the  button.</p> <p>There, enter "Dropdown_Hierarchies" as the Name.</p> <p>We will load the hierarchies into this Dropdown list later from a script.</p>	 <p>Show Styling Options For Dropdown</p> <p>▼ Application Design Properties</p> <p>Name Dropdown_Hierarchies</p>
<p>To make it clear what the contents of our Dropdown widgets are, we will insert a Label for each of the Dropdown widgets.</p> <p>To do that, click on the "+" icon in the <i>Insert</i> panel and select Text.</p>	

<p>Place the inserted Text widget to the left side of each Dropdown widget and select the first one to edit its properties. Go to the <i>Styling</i> panel and enter “Dropdown_Dimensions_Label” as the Name.</p>	 <p>The screenshot shows the Styling panel for a Text widget. At the top, there are two tabs: a blue one with a 'X' icon and an orange one labeled 'Styling'. Below the tabs, it says 'Show Styling Options For Text'. Under the 'Styling' tab, there's a section titled 'Application Design Properties' with a 'Name' field containing 'Dropdown_Dimensions_Label'.</p>
<p>To edit what the label shows, double click on the Text widget in the Canvas and enter “Dimension”.</p>	 <p>A screenshot of a Text widget on a canvas. The word "Dimension" is written inside it, and the entire text box is highlighted with a thick orange border.</p>
<p>Select the second label and go to the <i>Styling</i> panel and enter “Dropdown_Hierarchies_Label” as the Name.</p>	 <p>The screenshot shows the Styling panel for a Text widget. At the top, there are two tabs: a blue one with a 'X' icon and an orange one labeled 'Styling'. Below the tabs, it says 'Show Styling Options For Text'. Under the 'Styling' tab, there's a section titled 'Application Design Properties' with a 'Name' field containing 'Dropdown_Hierarchies_Label'.</p>
<p>To edit what the label shows, double click on the Text widget in the Canvas and enter “Hierarchies”.</p>	 <p>A screenshot of a Text widget on a canvas. The word "Hierarchies" is written inside it, and the entire text box is highlighted with a thick orange border.</p>
<p>We will now add the popup window that will display extra information about the selected measure, dimension, or data cell.</p> <p>To add a popup window, go to Popups in the Layout and click on the “+” icon next to it.</p>	 <p>The screenshot shows the 'Popups' section in the Layout panel. It features a dropdown menu with a downward arrow icon, a small icon of a window, and the word 'Popups'. To the right of the text is a blue '+' icon.</p>

<p>Click on the newly added popup and go to the <i>Styling</i> panel. There, enter “Popup_Details” in the Name input form and set the Popup Size to the width of 800 px and height of 400 px.</p>	
<p>Now, switch over to the <i>Builder</i> panel by clicking on the  button. Insert “Details” as the Title. Here, we will also enable the header and footer by toggling the button to Yes.</p>	
<p>We also want to add a button through which the user can exit the popup. Firstly, delete the buttons that can be found there by selecting each of them  and clicking on the  icon. To add this new button, click on the “+” icon next to Buttons.</p> <p>Insert the ID “BTN_Cancel”, the text “Cancel”, and check the options “Emphasized”, “Enabled”, and “Visible”. Click on Apply to save the changes.</p>	

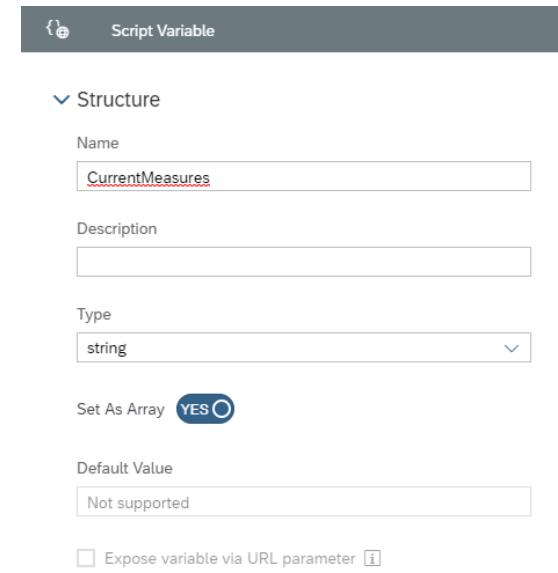
<p>To be able to display the extra information that we want in the popup, we need to add a chart. To do that, select the Chart icon from the <i>Insert</i> panel.</p>	
<p>Click on the Chart and go to the Designer to set its properties. First, go to the <i>Styling</i> panel and enter "Details_Chart" as the Name.</p>	 <p>Show Styling Options For Chart</p> <p>▼ Application Design Properties</p> <p>Name Details_Chart</p>
<p>Then, switch over to the <i>Builder</i> panel. There, select the data source <i>BestRun_Advanced</i>. Select Trend (Time Series) as the Chart Structure. Add Gross Margin, Gross Margin % Dev, Gross Margin abs Dev, and Gross Margin Plan as the Measures. Under Time, add Time as the dimension</p>	 <p>Builder</p> <p>Data Source BestRun_Advanced</p> <p>▼ Chart Structure</p> <p>Trend (selected)</p> <p>Comparison, Correlation, Indicator, More</p> <p>Measures</p> <ul style="list-style-type: none"> Gross Margin Gross Margin % Dev Gross Margin abs Dev Gross Margin Plan <p>+ Add Measure</p> <p>Time</p> <p>Time</p>
<p>To be able to access all the selections that the user made from any widget in our app, we need to add global variables. To add these script variables, go to Scripting and click the "+" next to Script Variables.</p>	<p>▼ Scripting</p> <p>▼ { Script Variables</p>

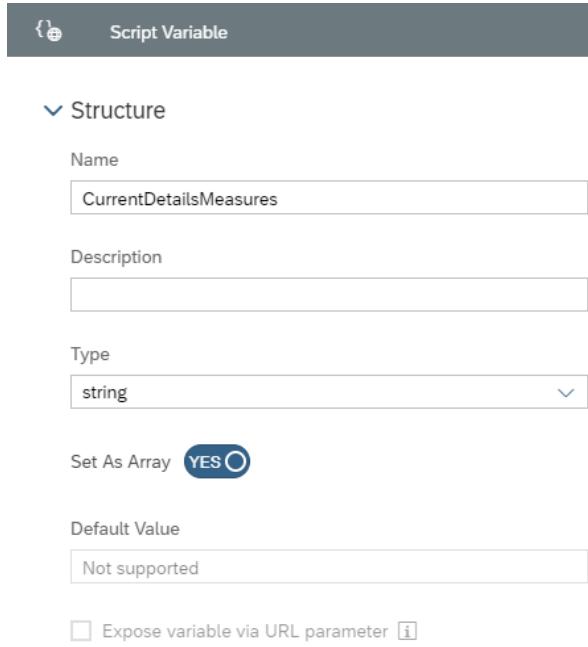
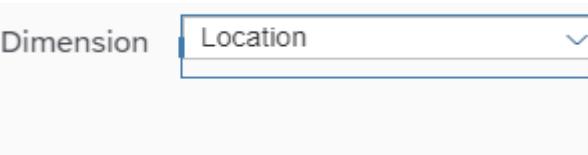
The first script variable we will add is one that will hold the current selection from the Dimensions Dropdown list. Add a script variable and in its properties enter "CurrentDimension" in the Name field, set the Type to "string", and the Default Value to "Location_4nm2e04531".



The second script variable we will add, is one that will hold the current measure selection(s) (Actual, Plan, Absolute, Percent).

Add a script variable and its properties enter "CurrentMeasures" in the Name field, set the Type to "string", and toggle the Set As Array button to "YES".



<p>The third, and last, script variable we will add will hold the data about the selections made that will be used to display the data in the popup window. Set "CurrentDetailsMeasures" as the Name, the Type to string, and toggle the Set As Array button to YES.</p>	
<p>Now, we will decide what will happen when a Dropdown list element in the Canvas is selected. Firstly, we will write the script for the first widget, the Dimensions Dropdown list. To do this, select the Dimensions Dropdown list in the Canvas and click  on the icon that appears next to it.</p>	

This will open the `onSelect` event script of the Dropdown widget. Here, we will first get the selected element of the list.

We will then remove any already set dimensions in the Table and the Chart and add the newly selected dimension to them.

We will also add that dimension to our Details Chart (the one that we added to the Popup window).

Afterwards, we will write the filter information in the browser's console and save the selection in our script variable, `CurrentDimension`.

Then, to set the available hierarchies for the selected dimension, we loop through the available hierarchies of our data source in relation to the current dimension and then we push all the available hierarchies in the Dropdown list of the Hierarchies.

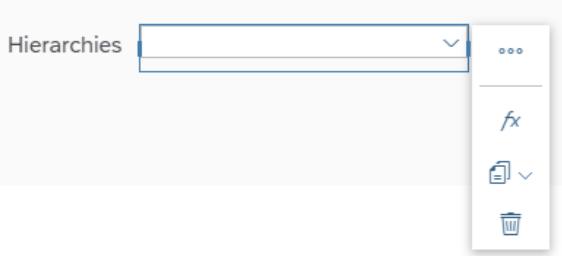
At the end, we set the default hierarchy of the Table, Chart, and Details Chart to Flat Presentation.

```
Canvas Dropdown_Dimensions - onSelect
Dropdown_Dimensions - onSelect
Called when the user selects an entry in the dropdown.
function onSelect() : void

1 var sel = Dropdown_Dimensions.getSelectedKey();
2
3 // Table
4 Table.removeDimension(CurrentDimension);
5 Table.addDimensionToRows(sel);
6
7 //Chart
8 Chart.removeDimension(CurrentDimension, Feed.CategoryAxis);
9 Chart.addDimension(sel, Feed.CategoryAxis);
10
11 //Details_Chart remove dimension filter
12 Details_Chart.getDataSource().removeDimensionFilter(CurrentDimension);
13
14 // write filter information into the browser console
15 console.log(['CurrentDimension: ', CurrentDimension]);
16 console.log(['Selection: ', sel]);
17
18 // save the current selection (dimension) into a global variable
19 CurrentDimension = sel;
20
21 // get hierarchies from the current dimension
22 var hierarchies = Table.getDataSource().getHierarchies(CurrentDimension);
23 var flag = true;
24
25 // remove all current items form the Dropdown_Hierarchies
26 Dropdown_Hierarchies.removeAllItems();
27
28 // loop
29 for (var i=0;i<hierarchies.length; i++){
30   if (hierarchies[i].id === '__FLAT__') {
31     Dropdown_Hierarchies.addItem(hierarchies[i].id, 'Flat Presentation');
32   }
33   else {
34     Dropdown_Hierarchies.addItem(hierarchies[i].id, hierarchies[i].description);
35     if (flag === true) {
36       var hierarchy = hierarchies[i].id;
37       flag = false;
38     }
39   }
40 }
41 // write hierarchy information to browser console
42 console.log(['Hierarchy: ', hierarchy]);
43 console.log(['Current Dimension: ', CurrentDimension]);
44
45 // set Flat Hierarchy als Default
46 Dropdown_Hierarchies.setSelectedKey('__FLAT__');
47
48 // Table
49 Table.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');
50
51 // Chart
52 Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');
53
54 // Details_Chart
55 Details_Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');

var sel = Dropdown_Dimensions.getSelectedKey();
// Table
Table.removeDimension(CurrentDimension);
Table.addDimensionToRows(sel);
// Chart
Chart.removeDimension(CurrentDimension,
Feed.CategoryAxis);
Chart.addDimension(sel, Feed.CategoryAxis);
// Details_Chart remove dimension filter
Details_Chart.getDataSource().removeDimensionFilter(C
urrentDimension);

// write filter information into the browser console
console.log(['CurrentDimension: ',
CurrentDimension]);
console.log(['Selection: ', sel]);
// save the current selection (dimension) into a
global variable
CurrentDimension = sel;
// get hierarchies from the current dimension
var hierarchies =
Table.getDataSource().getHierarchies(CurrentDimension
);
var flag = true;
// remove all current items form the
Dropdown_Hierarchies
Dropdown_Hierarchies.removeAllItems();
```

	<pre>// loop for (var i = 0; i < hierarchies.length; i++) { if (hierarchies[i].id === '__FLAT__') { Dropdown_Hierarchies.addItem(hierarchies[i].id, 'Flat Presentation'); } else { Dropdown_Hierarchies.addItem(hierarchies[i].id, hierarchies[i].description); if (flag === true) { var hierarchy = hierarchies[i].id; flag = false; } } } // write hierarchy information to browser console console.log(['Hierarchy: ', hierarchy]); console.log(['Current Dimension: ', CurrentDimension]); // set Flat Hierarchy as Default Dropdown_Hierarchies.setSelectedKey('__FLAT__'); // Table Table.getDataSource().setHierarchy(CurrentDimension, '__FLAT__'); // Chart Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__'); // Details_Chart Details_Chart.getDataSource().setHierarchy(CurrentDim ension, '__FLAT__');</pre>
<p>Now to edit the <code>onSelect</code> event script of the second Dropdown list, Hierarchies, select it in the Canvas and click on the  icon next to it.</p>	
<p>In the <code>onSelect</code> event script of this widget, we will simply set the hierarchy of the Table, Chart, and Display Chart (the one in the popup window) to the selected element of the Dropdown list.</p>	<p>Canvas Dropdown_Hierarchies - onSelect</p> <pre>Dropdown_Hierarchies - onSelect Called when the user selects an entry in the dropdown. function onSelect() : void 1 var sel = Dropdown_Hierarchies.getSelectedKey(); 2 3 // set hierarchy for Table 4 Table.getDataSource().setHierarchy(CurrentDimension, sel); 5 6 // set hierarchy for Chart 7 Chart.getDataSource().setHierarchy(CurrentDimension, sel); 8 9 // set hierarchy for Details Chart 10 Details_Chart.getDataSource().setHierarchy(CurrentDimension, sel); var sel = Dropdown_Hierarchies.getSelectedKey(); // set hierarchy for Table Table.getDataSource().setHierarchy(CurrentDimension, sel); // set hierarchy for Chart Chart.getDataSource().setHierarchy(CurrentDimension, sel); // set hierarchy for Details Chart Details_Chart.getDataSource().setHierarchy(CurrentDim ension, sel);</pre>

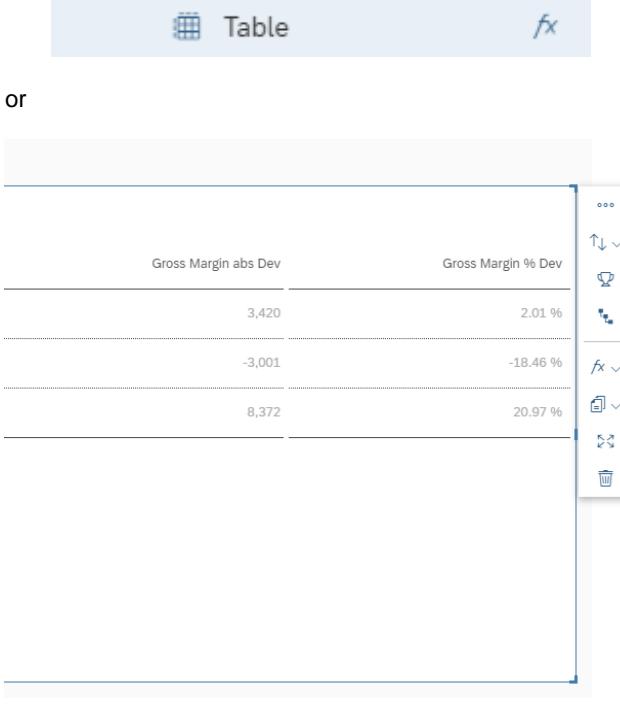
This use case enables the user to get more information about three things. A selected dimension, a selected measure, and a selected dimension, and a selected data cell. These selections can be made in the Table as well as the Chart.

We will start off by writing the script of the Table.

Open the `onSelect` event script of the Table by either selecting it in the Layout

or the Canvas and clicking on the  icon that appears next to it.

or



Gross Margin abs Dev	Gross Margin % Dev
3,420	2.01 %
-3,001	-18.46 %
8,372	20.97 %

In the `onSelect` event script of the Table we want to capture the selection made on the Table. We will write it into our console so that we can track the selections made.

We will set the visibility of the popup to false until we determine what the selected element was.

Afterwards, we will loop over the captured selected object of the Table and get whether it was a measure, a dimension, or a data cell (crossover between measure and dimension).

After capturing this information, we will push it unto the Chart in the popup window

We will then save the selected measures in the script variable `CurrentDetailsMeasures`.

Finally, we set the visibility of the popup to true which is then used to open it.

```

Canvas Table - onSelect
Table - onSelect
Called when the user makes a selection within the table.
function onSelect() : void

1 var sel = Table.getSelections();
2
3 console.log( ['Table Selection: ', sel]);
4
5 Details_Chart.getDataSource().removeDimensionFilter(CurrentDimension);
6
7 var Popup_show = false;
8
9 if (sel.length > 0) {
10    var selection = sel[0];
11
12    for (var dimensionId in selection) {
13        var memberId = selection[dimensionId];
14
15        if (dimensionId === '@MeasureDimension') {
16            // Measure
17            console.log( ['Selection Measure: ', dimensionId]);
18            console.log( ['Selection Member: ', memberId]);
19
20            // remove current measure
21            console.log( ['CurrentMeasures: ', CurrentMeasures]);
22
23            for (var i=0;i<CurrentMeasures.length; i++) {
24                Details_Chart.removeMeasure(CurrentMeasures[i], Feed.ValueAxis);
25                Details_Chart.addMeasure(memberId, Feed.ValueAxis);
26            }
27
28        // Details_Chart.addMeasure(memberId, Feed.ValueAxis);
29        CurrentDetailsMeasures.push(memberId);
30        Popup_show = true;
31
32    }
33    // Dimension
34    else {
35        console.log( ['Selection Dimension: ', dimensionId]);
36        console.log( ['Selection Member: ', memberId]);
37
38        Details_Chart.getDataSource().setDimensionFilter(dimensionId, memberId);
39        Popup_show = true;
40    }
41 }
42
43
44
45 if (Popup_show === true) {
46    Popup_Details.open();
47 }
48
49
50
51
52
53

```

```

var sel = Table.getSelections();
console.log(['Table Selection: ', sel]);
Details_Chart.getDataSource().removeDimensionFilter(C
urrentDimension);
var Popup_show = false;
if (sel.length > 0) {
    var selection = sel[0];

    for (var dimensionId in selection) {
        var memberId = selection[dimensionId];

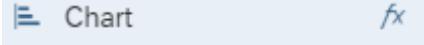
        if (dimensionId === '@MeasureDimension') {
            // Measure
            console.log(['Selection Measure: ',
dimensionId]);
            console.log(['Selection Member: ', memberId]);
            // remove current measure
            console.log(['CurrentMeasures: ',
CurrentMeasures]);

            for (var i = 0; i < CurrentMeasures.length;
i++) {

                Details_Chart.removeMeasure(CurrentMeasures[i],
Feed.ValueAxis);
                Details_Chart.addMeasure(memberId,
Feed.ValueAxis);
            }

            // Details_Chart.addMeasure(memberId,
Feed.ValueAxis);
            CurrentDetailsMeasures.push(memberId);
        }
    }
}

```

	<pre>Popup_show = true; } // Dimension else { console.log(['Selection Dimension: ', dimensionId]); console.log(['Selection Member: ', memberId]); Details_Chart.getDataSource().setDimensionFilter(dime nsionId, memberId); Popup_show = true; } } if (Popup_show === true) { Popup_Details.open(); }</pre>
<p>Now, we need to do the same for the Chart.</p> <p>Opposed to the Table, in the Chart, the user can only click on a dimension and can click on the chart bars which are crossovers of a measure and a dimension.</p> <p>To write the script of the Chart, select the widget in the Layout and click on the  icon next to it.</p>	

In the `onSelect` event script of the Chart, we will get the selected element of the Chart and save it in a local variable, sel.

We will set the popup window's visibility to false and remove the current measures from the Details_Chart in the popup.

And then, if it's a measure, we will add it as a measure to the Details_Chart and if it's a dimension, we will set it as a dimension filter of the Details_Chart.

We will then push the selected measures, if any, unto the script variable CurrentDetailsMeasures.

At the end, the popup window's visibility is set to true and is opened.

```

Canvas ▾ Chart - onSelect ×
Chart - onSelect
Called when the user makes a selection within the chart.
function onSelect() : void

1 var sel = Chart.getSelections();
2 console.log(['Chart Selection: ', sel, CurrentMeasures]);
3 var Popup_show = false;
4
5 if (sel.length > 0) {
6
7   Details_Chart.getDataSource().removeDimensionFilter(CurrentDimension);
8
9   // remove the current measures
10  for (var i=0;i<CurrentMeasures.length; i++) {
11    Details_Chart.removeMeasure(CurrentMeasures[i], Feed.ValueAxis);
12  }
13
14  for (i=0;i<sel.length; i++) {
15
16    var selection = sel[i];
17
18    for (var dimensionId in selection) {
19      var memberId = selection[dimensionId];
20
21      if (dimensionId === '@MeasureDimension') {
22        // Measure
23        console.log(['Add Selection Measure: ', dimensionId]);
24        console.log(['Add Selection Member: ', memberId]);
25
26        Details_Chart.addMeasure(memberId, Feed.ValueAxis);
27        CurrentDetailsMeasures.push(memberId);
28        Popup_show = true;
29      }
30      // Dimension
31    else {
32      console.log(['Selection Dimension: ', dimensionId]);
33      console.log(['Selection Member: ', memberId]);
34
35      Details_Chart.getDataSource().setDimensionFilter(dimensionId, memberId);
36      Popup_show = true;
37    }
38  }
39 }
40 }
41
42 if (Popup_show === true) {
43   Popup_Details.open();
44 }
45
46
47

var sel = Chart.getSelections();
console.log(['Chart Selection: ', sel,
CurrentMeasures]);
var Popup_show = false;

if (sel.length > 0) {

  Details_Chart.getDataSource().removeDimensionFilter(C
urrentDimension);

  // remove the current measures
  for (var i = 0; i < CurrentMeasures.length; i++) {
    Details_Chart.removeMeasure(CurrentMeasures[i],
Feed.ValueAxis);
  }

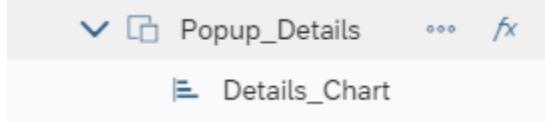
  for (i = 0; i < sel.length; i++) {
    var selection = sel[i];

    for (var dimensionId in selection) {
      var memberId = selection[dimensionId];

      if (dimensionId === '@MeasureDimension') {
        // Measure
        console.log(['Add Selection Measure: ',
dimensionId]);
        console.log(['Add Selection Member: ',
memberId]);

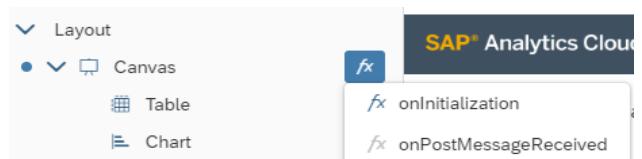
        Details_Chart.addMeasure(memberId,
Feed.ValueAxis);
        CurrentDetailsMeasures.push(memberId);
        Popup_show = true;
      }
    }
  }
}

```

	<pre> // Dimension else { console.log(['Selection Dimension: ', dimensionId]); console.log(['Selection Member: ', memberId]); } Details_Chart.getDataSource().setDimensionFilter(dimensionId, memberId); Popup_show = true; } } if (Popup_show === true) { Popup_Details.open(); } </pre>
<p>In previous steps, we had created the popup window and added a Cancel button. To make the button do anything, we need to write a script for it.</p> <p>To do that, select <code>Popup_Details</code> in the Layout and click on the  next to it.</p>	
<p>This will open the <code>onButtonClick</code> event script of the <code>Popup</code> widget. Here, we will set what happens when the user clicks on the Cancel button.</p> <p>Firstly, we will remove the content, if there is any, of the <code>CurrentDetailsMeasures</code> from the <code>Details_Chart</code> and set the default measures, from the <code>CurrentMeasures</code> script variable, as the measures of the <code>Details_Chart</code>.</p> <p>At the end, we will trigger the closing of the popup.</p>	<pre> Popup_Details - onButtonClick Called when the user clicks any button in the footer of the popup. function onButtonClick(buttonId: string) : void 1 // remove the current measure selection and set all default measures for the details chart 2 3 for (var i=0;i<CurrentDetailsMeasures.length; i++){ 4 Details_Chart.removeMeasure(CurrentDetailsMeasures[i], Feed.ValueAxis); 5 } 6 7 CurrentDetailsMeasures = ArrayUtils.create(Type.string); 8 9 for (i=0;i<CurrentMeasures.length; i++){ 10 Details_Chart.addMeasure(CurrentMeasures[i], Feed.ValueAxis); 11 } 12 13 // close the popup 14 Popup_Details.close(); // remove the current measure selection and set all default measures for the details chart for (var i = 0; i < CurrentDetailsMeasures.length; i++) { Details_Chart.removeMeasure(CurrentDetailsMeasures[i], Feed.ValueAxis); } CurrentDetailsMeasures = ArrayUtils.create(Type.string); for (i = 0; i < CurrentMeasures.length; i++) { Details_Chart.addMeasure(CurrentMeasures[i], Feed.ValueAxis); } // close the popup Popup_Details.close(); </pre>

The last script we will write is the one for the Canvas. This script gets executed on the initialization of the Canvas.

Select the Canvas element in the Layout and click on the  icon next to it and select onInitialization.



In this script, we will load the hierarchies into the Hierarchies Dropdown list and set the default hierarchy to Flat Presentation.

At the end, we will also fill the script variable CurrentMeasures with the available measures of Gross margin (Actual, Plan, Absolute, and Percent)

```
Canvas Application - onInitialization X
Application - onInitialization
Called when the analytic application has finished loading.
function onInitialization() : void

// get hierarchies from the current dimension
var hierarchies = Table.getDataSource().getHierarchies(CurrentDimension);
var flag = true;
for (var i=0;i<hierarchies.length; i++) {
    if (hierarchies[i].id === '__FLAT__') {
        Dropdown_Hierarchies.addItem(hierarchies[i].id, 'Flat Presentation');
    }
    else {
        Dropdown_Hierarchies.addItem(hierarchies[i].id, hierarchies[i].description);
        if (flag === true) {
            var hierarchy = hierarchies[i].id;
            flag = false;
        }
    }
}
// write hierarchy information to browser console
console.log( ['Hierarchy: ', hierarchy]);
console.log( ['Current Dimension: ', CurrentDimension]);
// set Flat Hierarchy als Default
Dropdown_Hierarchies.setSelectedKey('__FLAT__');

//Table
Table.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');

//Chart
Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');

//Details_Chart
Details_Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');

//fill global Variable CurrentMeasures
CurrentMeasures.push('[Account_BestRunJ_sold].[parentId].[GrossMarginActual]');
CurrentMeasures.push('[Account_BestRunJ_sold].[parentId].[GrossMarginPlan]');
CurrentMeasures.push('[Account_BestRunJ_sold].[parentId].[GrossMarginAbs]');
CurrentMeasures.push('[Account_BestRunJ_sold].[parentId].[GrossMarginPercent]');

// get hierarchies from the current dimension
var hierarchies =
Table.getDataSource().getHierarchies(CurrentDimension );
var flag = true;

// loop
for (var i = 0; i < hierarchies.length; i++) {
    if (hierarchies[i].id === '__FLAT__') {
        Dropdown_Hierarchies.addItem(hierarchies[i].id,
'Flat Presentation');
    }
    else {
        Dropdown_Hierarchies.addItem(hierarchies[i].id,
hierarchies[i].description);
        if (flag === true) {
            var hierarchy = hierarchies[i].id;
            flag = false;
        }
    }
}
// write hierarchy information to browser console
console.log(['Hierarchy: ', hierarchy]);
console.log(['Current Dimension: ',
CurrentDimension]);

// set Flat Hierarchy as Default
Dropdown_Hierarchies.setSelectedKey('__FLAT__');

// Table
Table.getDataSource().setHierarchy(CurrentDimension,
 '__FLAT__');

// Chart
Chart.getDataSource().setHierarchy(CurrentDimension,
 '__FLAT__');
```

```
// Details_Chart
Details_Chart.getDataSource().setHierarchy(CurrentDimension, '__FLAT__');

// fill global Variable CurrentMeasures
CurrentMeasures.push('[Account_BestRunJ_sold].[parent
Id].&[Gross_MarginActual]');
CurrentMeasures.push('[Account_BestRunJ_sold].[parent
Id].&[Gross_MarginPlan]');
CurrentMeasures.push('[Account_BestRunJ_sold].[parent
Id].&[Gross_Margin_Abs]');
CurrentMeasures.push('[Account_BestRunJ_sold].[parent
Id].&[Gross_Margin_Percent]');
```

Now let's see how it looks like.

Click on Run Analytic Application in the upper right side of the page and the result should look something like this:

If we click on one of the dimension data cells, in this example the dimension is set to Location and we clicked on Los Angeles, the popup window will appear. It gives us an overview of all the measures (Gross Margin Actual, Plan, Absolute, Percent) in relation to the selected Location (Los Angeles) over the Time factor.

When opening the browser's console, we can also see that the selection was printed there.

If we click on one of the measures, in this screenshot we chose Gross Margin Actual, the measure is shown in the popup window in relation to the Time factor.

The selection is also printed out in the console:

The third option to select in the Table is an individual data cell.

In this example, we changed the Dimension to store and selected the data cell at the crossover of Gross Margin Plan and Country Fair Foods. This triggers the opening of a popup window that shows Gross Margin Plan in relation with Time and with a Store Filter of Country Fair Foods.

The selection triggers the following console message:

If we change the dimension back to Location and change the hierarchy of the Table to States, the following Table will be displayed:

We can then choose a state and we would also get a popup window that displays the measures in regard to a state (here, Nevada was selected).

This state selection prints the following message to the browser console:

Now, we will look at how the Chart behaves.

To switch to the Chart, click on the



icon in the Canvas.

There, we will firstly click on a dimension. Here, the dimension filter was set to Product and Lemonade was clicked on.

	Gross Margin Plan	Gross Margin	Gross Margin abs Dev
Los Angeles	65,012	65,012	-65
Reno	6,083	6,083	-616
Henderson	2,322	2,322	-4719
Carson City	515	515	-23
Portland	10,495	10,495	-1,402
Salem	14,680	14,680	-6,607
Eugene	8,934	8,934	1,385
Gresham	4,483	4,483	2,204
Hillsboro	820	820	263
Beaverton	4,088	4,088	-1,710
San Francisco	21,491	21,491	-2,711
San Diego	17,002	17,002	3,950
Sacramento	24,059	24,059	3,374
San Jose	24,802	24,802	-2,857
Oakland	12,754	12,754	78
Santa Barbara	11,174	11,174	1,408
Beverly Hills	13,760	13,760	-365
Las Vegas	4,234	4,234	-443



▶ (2) ["Table Selection: ", Array(1)]

sand!

▶ (2) ["Selection Dimension: ", "Location_4nm2e04531"]

▶ (2) ["Selection Member: ", "CT1"]

sand!

sand!



▶ (2) ["Selection Measure: ", "@MeasureDimension"]

sandbox.worker.main..81de1d8c7a541.js:22

▶ (2) ["Selection Member: ", "[Account_BestRun_sold].[parentID].@{[GrossMarginActual]}"]

sandbox.worker.main..81de1d8c7a541.js:22



Cancel

All the measures are shown in regard to Time and with a product filter of Lemonade.

This selection prints the following message into the console:
The measures are added according to the chosen product

The second thing we can click on in the Chart is a specific measure in regard to a specific dimension.
For example, Gross Margin Abs Dev in relation to Orange with pulp (the chart bar marked in the screenshot).

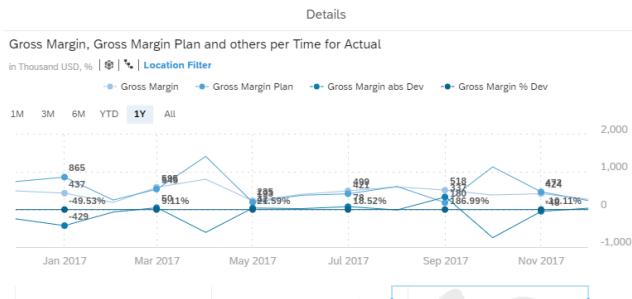
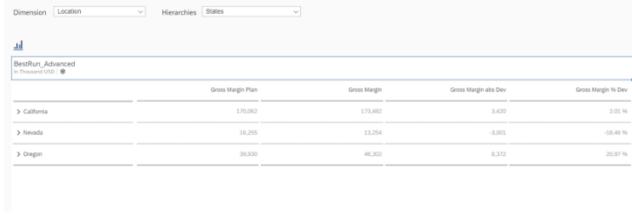
This causes a popup window to appear that displays the measure chosen (Gross Margin abs Dev) per Time and with a dimension filter (here: Product – Orange with pulp)

This triggers the printing of the following messages in the browser's console:

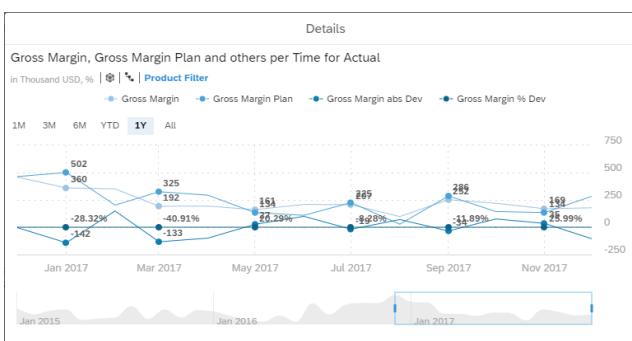
Note: The user can always check what filter is being used by clicking on Filter.

This opens a list of filters used – here only one product (Orange with pulp) has been used as a filter.

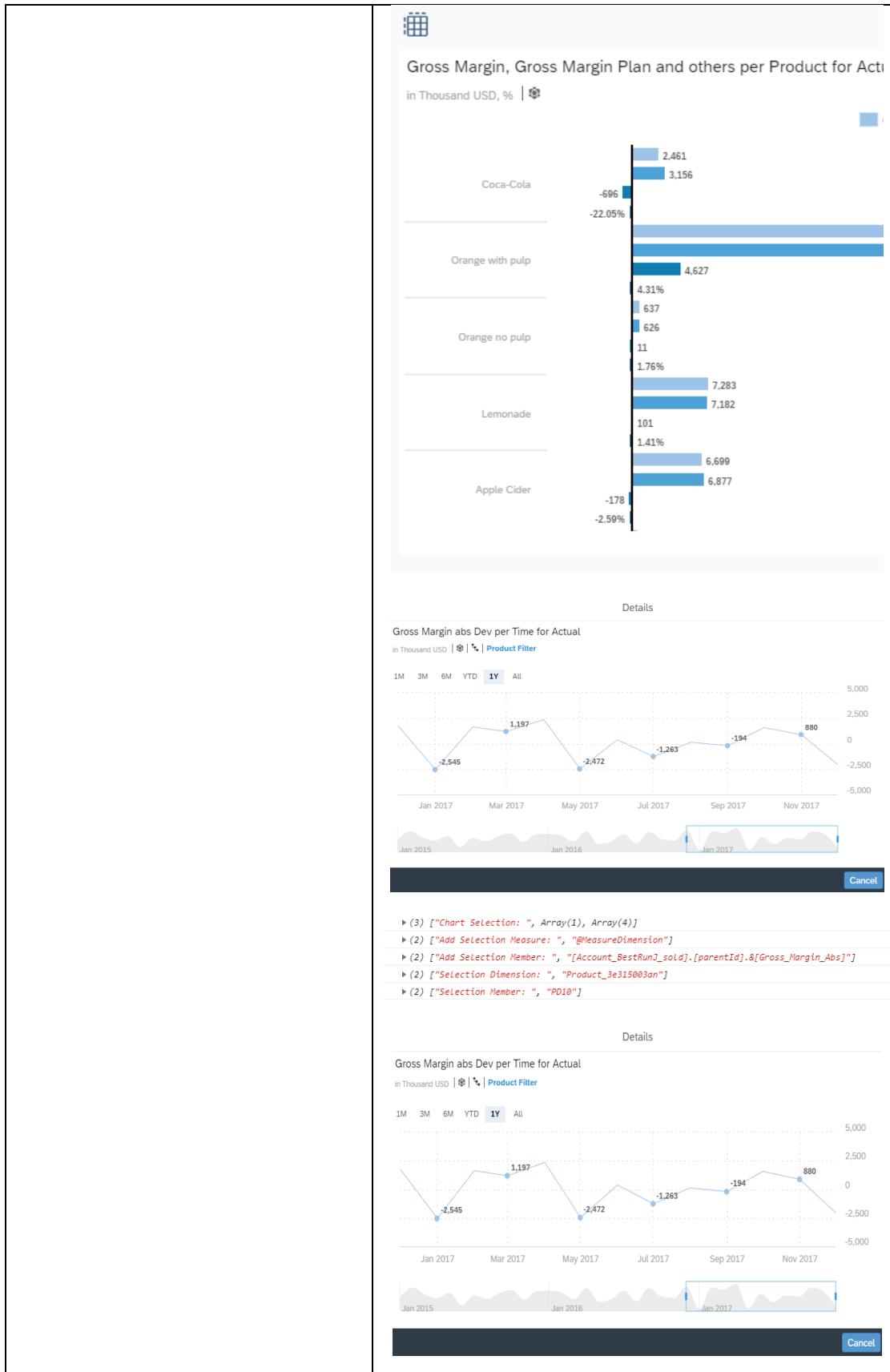
```
> (2) ["Selection Measure: ", "@MeasureDimension"]
      sandbox.worker.main..81de1d8c7a541.js:22
> (2) ["Selection Member: ", "[Account_BestRun_sold].[parentId].&[GrossMarginPlan]"]
      sandbox.worker.main..81de1d8c7a541.js:22
> (2) ["CurrentMeasures: ", Array(4)]
      sandbox.worker.main..81de1d8c7a541.js:22
> (2) ["Selection Dimension: ", "Store_3z2g5g06m4"]
      sandbox.worker.main..81de1d8c7a541.js:22
> (2) ["Selection Member: ", "ST103"]
      sandbox.worker.main..81de1d8c7a541.js:22
```

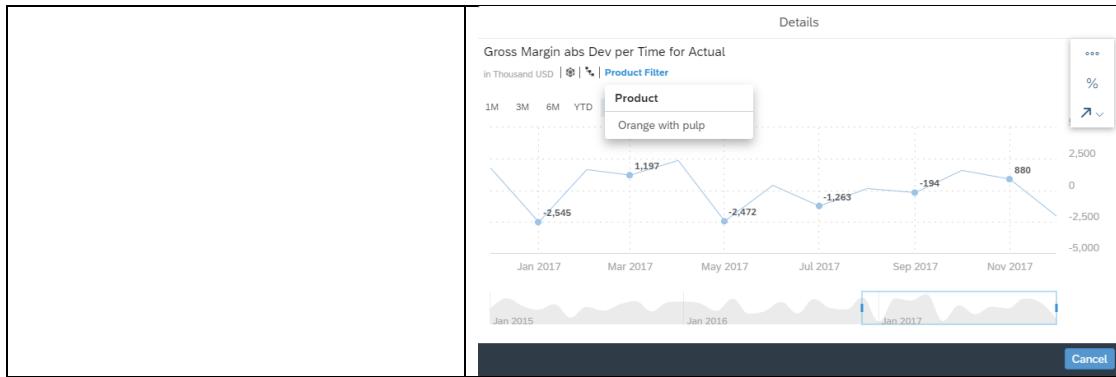


```
> (2) ["Table Selection: ", Array(1)]
> (2) ["Selection Dimension: ", "Location_4nm2e04531"]
> (2) ["Selection Member: ", "[Location_4nm2e04531].[States].&[SA2]"]
```



```
> (3) ["Chart Selection: ", Array(4), Array(4)]
> (2) ["Add Selection Measure: ", "@MeasureDimension"]
> (2) ["Add Selection Member: ", "[Account_BestRun_sold].[parentId].&[GrossMarginActual]"]
> (2) ["Selection Dimension: ", "Product_3e315003an"]
> (2) ["Selection Member: ", "PD12"]
> (2) ["Add Selection Measure: ", "@MeasureDimension"]
> (2) ["Add Selection Member: ", "[Account_BestRun_sold].[parentId].&[GrossMarginPlan]"]
> (2) ["Selection Dimension: ", "Product_3e315003an"]
> (2) ["Selection Member: ", "PD12"]
> (2) ["Add Selection Measure: ", "@MeasureDimension"]
> (2) ["Add Selection Member: ", "[Account_BestRun_sold].[parentId].&[GrossMarginAbs]"]
> (2) ["Selection Dimension: ", "Product_3e315003an"]
> (2) ["Selection Member: ", "PD12"]
> (2) ["Add Selection Measure: ", "@MeasureDimension"]
> (2) ["Add Selection Member: ", "[Account_BestRun_sold].[parentId].&[GrossMarginPercent]"]
> (2) ["Selection Dimension: ", "Product_3e315003an"]
> (2) ["Selection Member: ", "PD12"]
```





6.9 Using R Widget Word Cloud for Visualization

This application features an overview for the customer complaints a company got from its customers over the years 2018 and 2019.

In the Canvas, we will add a Table with our top 10 customers as well as a Chart with the complaints of the customers. Other than that, we will have two R Visualization widgets through which we will create word clouds that change the size of the words displayed according to the frequency with which they appear in the data set.

Further functionalities in this application include how to filter widgets according to a selected element of a Table and how we can change the color of the word clouds through external input (in this use case, it's achieved through a Radio Button Group that has a script that passes the value to the R widgets.)

And lastly, the filtering of all the widgets in the Canvas using Radio Button Groups will be explored (here, we will filter according to *Regions* and according to the selected *Region*, several countries from that *Region* will be displayed in another Radio Button Group (*Country*) for further filtering of the widgets).

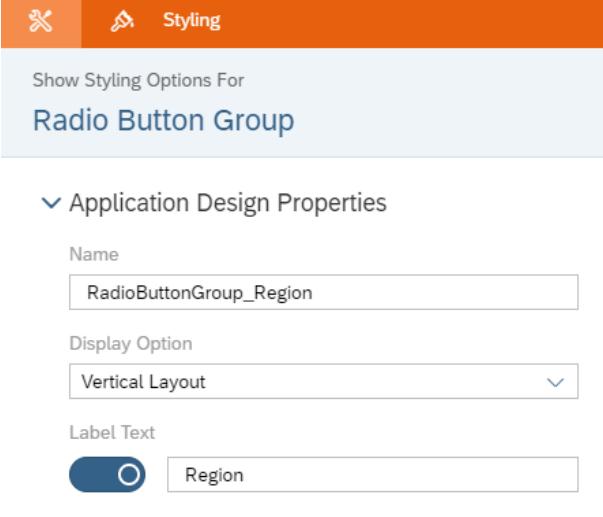
The result will look like this when we run the application:



Figure 72: Example Application Word Cloud

There are no prerequisites for this use case. You can start with a new application.

We recommend using the same names as that exercise for the used widgets so that the scripts in this use case don't have to be altered.

<p>The first thing we will do, is add a Radio Button Group to our Canvas where we will enable the user to choose between regions</p> <p>.</p>	 <p>+</p> <p>Dropdown</p> <p>checkbox icon</p> <p>checkbox icon</p> <p>radio icon</p> <p>radio icon</p> <p>button icon</p> <p>button icon</p> <p>filter line icon</p> <p>Filter Line</p>
<p>Select the newly added widget in the Canvas and go to the Designer (by clicking on Designer on the upper right side of the screen) and switch to the  Styling panel by clicking on the button.</p> <p>There, enter "RadioButtonGroup_Region" as the Name, choose Vertical Layout as the Display Option, and toggle the Label Text button to enable it and write "Region" as the Label Text.</p>	 <p>Styling</p> <p>Show Styling Options For Radio Button Group</p> <p>Application Design Properties</p> <p>Name: RadioButtonGroup_Region</p> <p>Display Option: Vertical Layout</p> <p>Label Text: Region</p>

Now, we will insert the options we want available in our Radio Button Group widget.

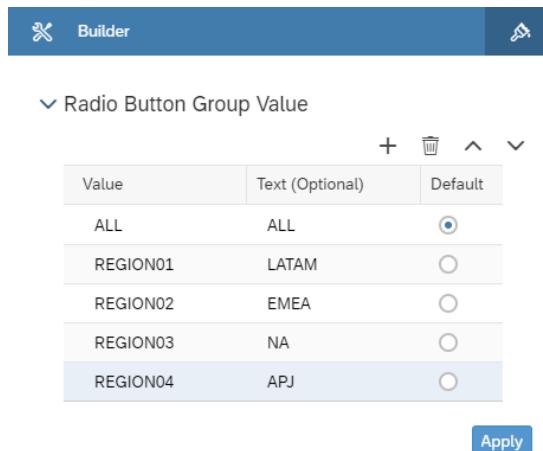
To do that, switch to the *Builder* panel by clicking on the  button.

Once there, start adding values using the “+” button.

We will add an option that has all the regions (1), and the others will be for Latin America (2), Europe, the Middle East and Africa (3), North America (4), and the Asia-Pacific region (5).

We will set All as our default value; this means that the widgets in our Canvas will be by default filtered according to that option and the user can change it afterwards.

Click on Apply to save the changes to your application.

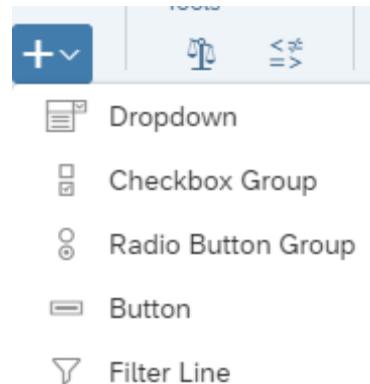


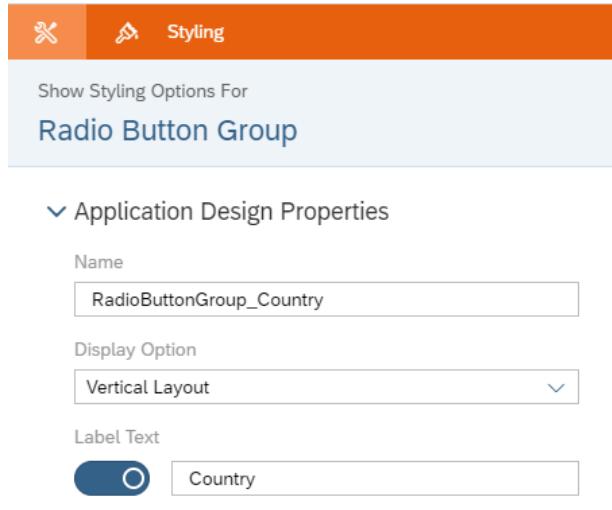
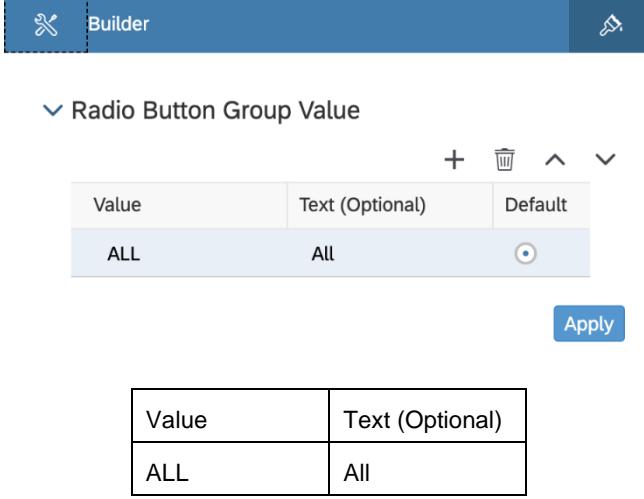
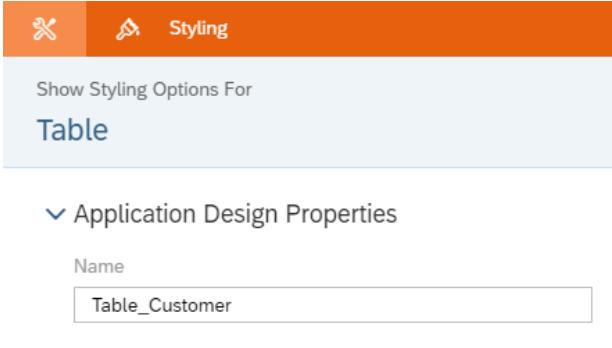
Value	Text (Optional)	Default
ALL	ALL	<input checked="" type="radio"/>
REGION01	LATAM	<input type="radio"/>
REGION02	EMEA	<input type="radio"/>
REGION03	NA	<input type="radio"/>
REGION04	APJ	<input type="radio"/>

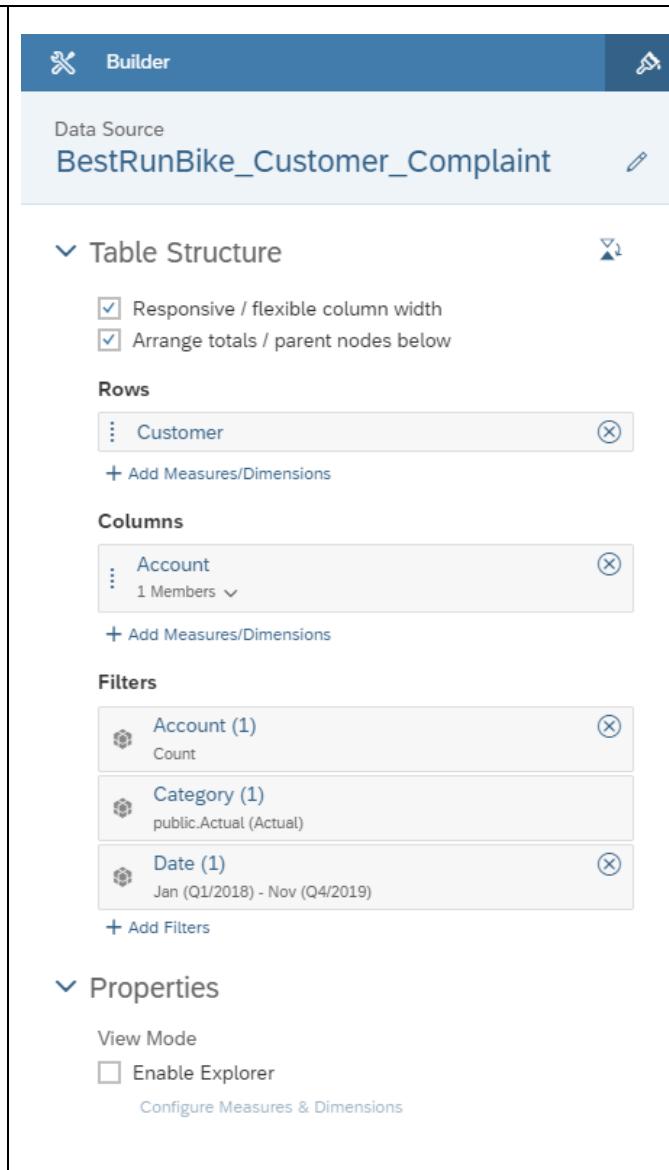
Value	Text (Optional)
ALL	ALL
REGION01	LATAM
REGION02	EMEA
REGION03	NA
REGION04	APJ

To enable further filtering, we will insert another Radio Button Group that houses the countries.

The values of this widget will change depending on the region selected. Place the widget underneath the Region Radio Button Group.

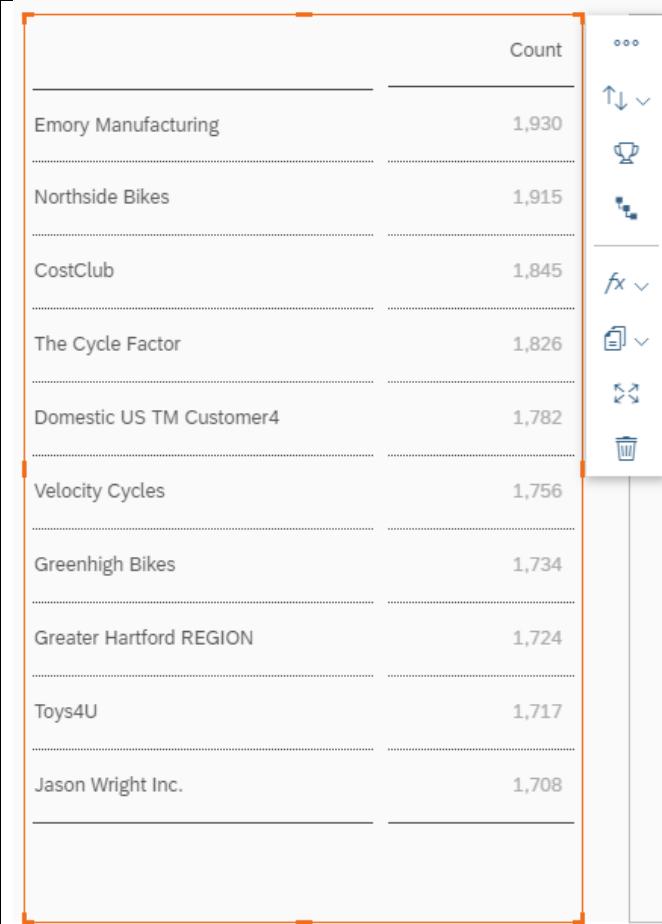


<p>Select the widget in the Canvas and go to the <i>Styling</i> panel.</p> <p>There, enter “RadioButtonGroup_Country” as the Name, choose Vertical Layout as the Display Option, and toggle the Label Text button to enable it and write “Country” as the Label Text.</p>							
<p>Now, we will insert the options we want available in our Radio Button Group widget.</p> <p>To do that, switch to the <i>Builder</i> panel by clicking on the  button.</p> <p>For this widget, we will only add one option which is “All”.</p> <p>To do that, click on the “+” button and add the values like in the screenshot to the right.</p> <p>Lastly, click on Apply to save the changes.</p> <p>(The other countries will be added through a script later in this tutorial.)</p>	 <table border="1" data-bbox="822 1181 1267 1291"> <thead> <tr> <th>Value</th><th>Text (Optional)</th><th>Default</th></tr> </thead> <tbody> <tr> <td>ALL</td><td>All</td><td><input checked="" type="radio"/></td></tr> </tbody> </table>	Value	Text (Optional)	Default	ALL	All	<input checked="" type="radio"/>
Value	Text (Optional)	Default					
ALL	All	<input checked="" type="radio"/>					
<p>Now, we will move on to add our Table, Chart, and R Widgets.</p> <p>We will start off with the Table.</p> <p>Through the <i>Insert</i> panel, add a new Table and place it to the right side of the two radio button groups.</p> <p>Select BestRunBike_Customer_Complaint as the data source.</p>							
<p>In the <i>Styling</i> panel of the Table insert the Name “Table_Customer”.</p>	 <table border="1" data-bbox="838 1956 1351 2001"> <tr> <td>Name</td> </tr> <tr> <td>Table_Customer</td> </tr> </table>	Name	Table_Customer				
Name							
Table_Customer							

<p>Afterwards, switch over to the <i>Builder</i> panel and there, enter the values as in the screenshot to the right.</p> <p>Check Responsive/flexible columns width</p> <p>Check Arrange totals/parent nodes below</p> <p>Add Customer to Rows</p> <p>Add Account to Columns</p> <p>And set the Filters to</p> <p>Account – Count</p> <p>Category – Actual</p> <p>Date – Jan (Q1/2018) – Nov (Q4/2019)</p> <p>This Table will hold the customers of our data set.</p>	 <p>The screenshot shows the Tableau Builder interface for the 'Customer' table. At the top, it says 'Data Source' and 'BestRunBike_Customer_Complaint'. Below that is the 'Table Structure' section, which includes checkboxes for 'Responsive / flexible column width' and 'Arrange totals / parent nodes below'. Under 'Rows', there is a box containing 'Customer' with a delete icon. A '+ Add Measures/Dimensions' button is also present. Under 'Columns', there is a box containing 'Account' with a dropdown menu showing '1 Members'. A '+ Add Measures/Dimensions' button is also present. Under 'Filters', there are three boxes: 'Account (1)' with 'Count', 'Category (1)' with 'public.Actual (Actual)', and 'Date (1)' with 'Jan (Q1/2018) - Nov (Q4/2019)'. Each filter has a delete icon. A '+ Add Filters' button is also present. Below the filters is the 'Properties' section, which includes 'View Mode' and a checkbox for 'Enable Explorer'. A 'Configure Measures & Dimensions' link is also present.</p>
--	--

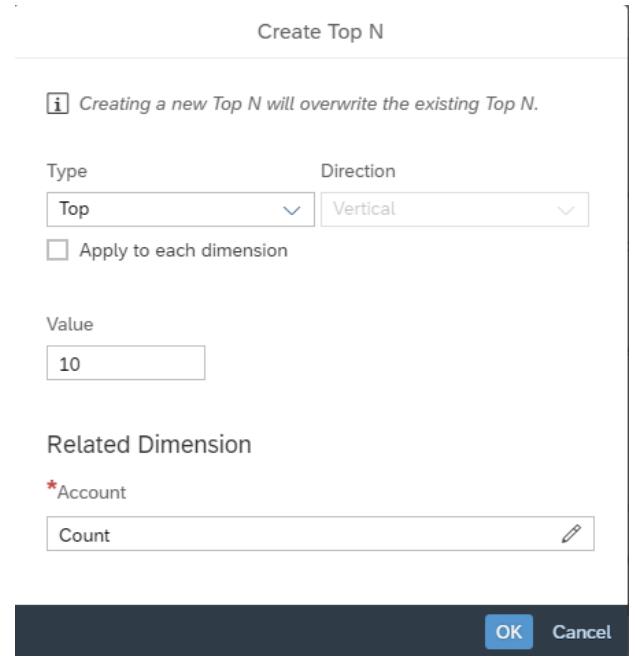
To have a better visual over our Customers and their complaints, we will only show the Top 10 Customers in our Table.

To do that, select the Table in the Canvas and click on the  icon in its menu.



	Count
Emory Manufacturing	1,930
Northside Bikes	1,915
CostClub	1,845
The Cycle Factor	1,826
Domestic US TM Customer4	1,782
Velocity Cycles	1,756
Greenhigh Bikes	1,734
Greater Hartford REGION	1,724
Toys4U	1,717
Jason Wright Inc.	1,708

This opens a “Create Top N” window. Enter “Top” as the Type, 10 as the Value, and Count in the Related Dimension’s Account field.



Create Top N

i Creating a new Top N will overwrite the existing Top N.

Type	Direction
Top	Vertical

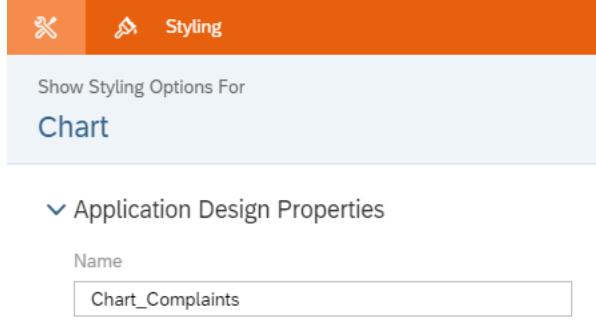
Apply to each dimension

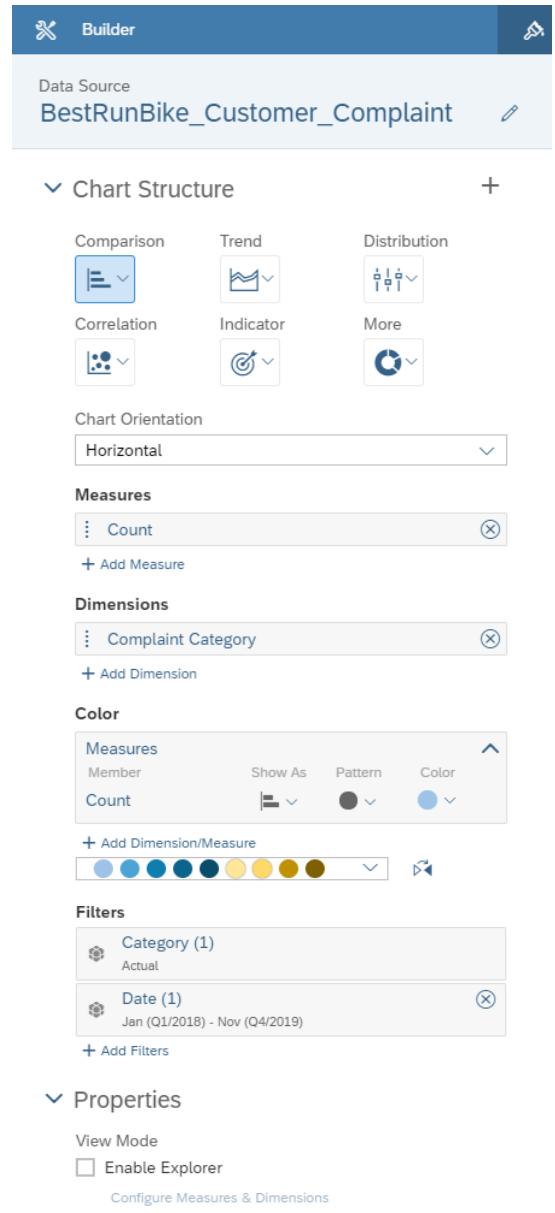
Value
10

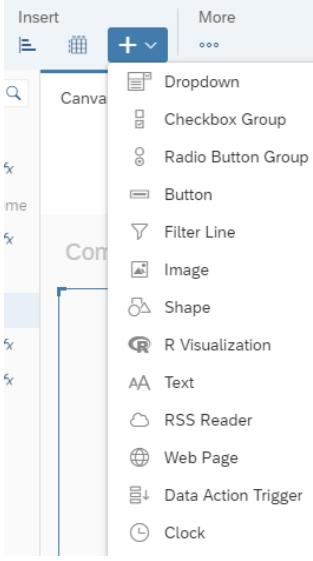
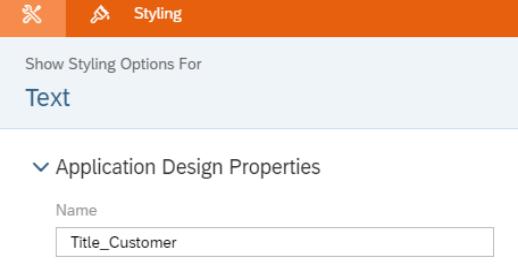
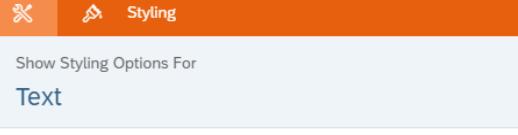
Related Dimension

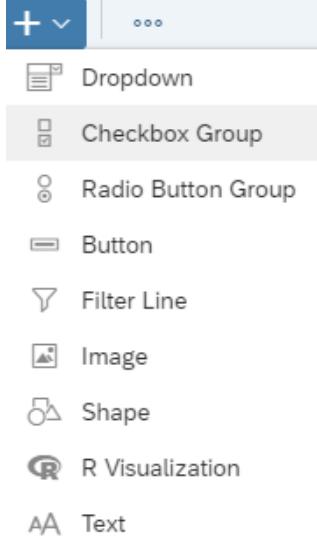
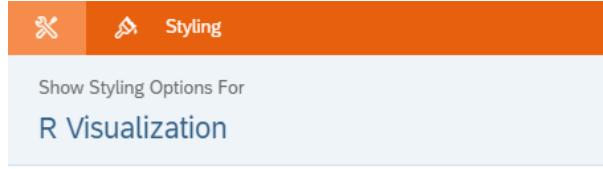
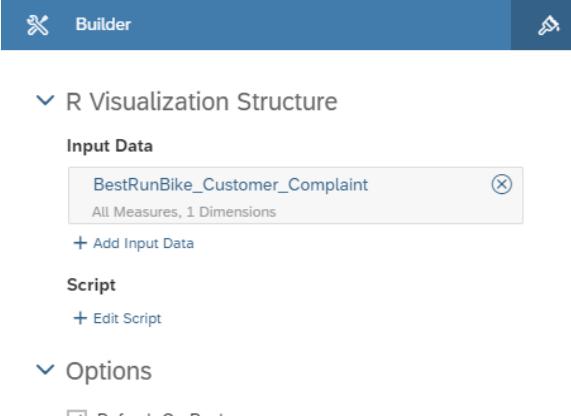
*Account
Count

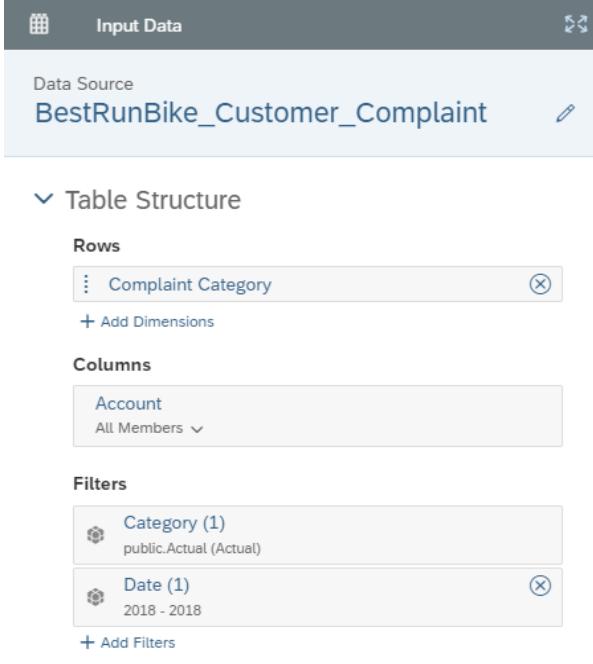
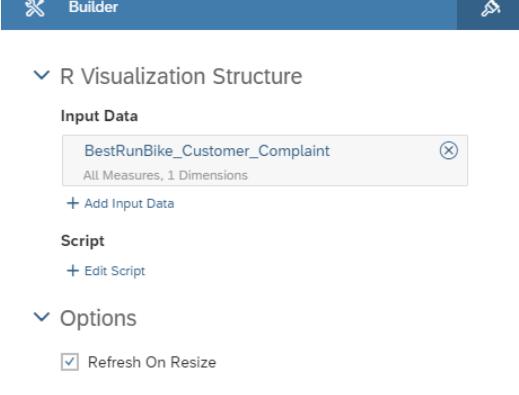
OK Cancel

<p>We also want to be able to view the complaints that we got from our customers, which is why we will add a chart to display them.</p> <p>First, we need to add the Chart. We will do that, again, through the <i>Insert</i> panel.</p>	
<p>To edit the properties of our Chart, go to the Designer.</p> <p>We'll change the <i>Styling</i> panel properties first.</p> <p>In the <i>Styling</i> panel enter "Chart_Complaints" as the Name.</p>	 <p>Show Styling Options For Chart</p> <p>Application Design Properties</p> <p>Name Chart_Complaints</p>

<p>To display the complaints of our customers, we will edit the <i>Builder</i> components of the Chart.</p> <p>Switch over to the Chart's <i>Builder</i> panel and enter the values as seen in the screenshot:</p> <p>Chart Structure: Comparison (Bar/Column) Chart Orientation: Horizontal Measures: Count Dimensions: Complaint Category Filters: Category: Actual Date: Jan (Q1/2018) – Nov (Q4/2019)</p>	 <p>The screenshot shows the Power BI Builder interface with the following configuration:</p> <ul style="list-style-type: none">Data Source: BestRunBike_Customer_ComplaintChart Structure: Comparison (selected)Measures: CountDimensions: Complaint CategoryColor: Measures (Count) set to Show As Bar, Pattern Solid, Color BlueFilters: Category (1) - Actual, Date (1) - Jan (Q1/2018) - Nov (Q4/2019)Properties: View Mode (checkbox checked), Enable Explorer (checkbox checked)
---	---

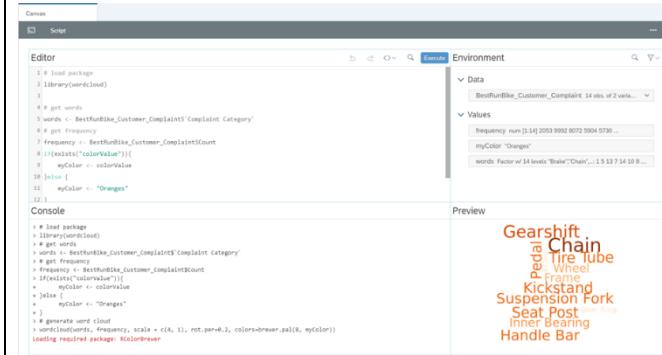
<p>To know what the Table and Chart represent, we will add text labels on top of each of them.</p> <p>Add two labels and place one above the Table and the other above the Chart.</p>	
<p>Click on the first text widget and open the <i>Styling</i> panel. There, enter “Title_Customer” as the Name.</p>	
<p>The previous step just edits the name through which the widget is mentioned if it's called in a script in the application. To edit what appears in the text box, double click on it in the Canvas and enter “Top 10 Customers” in the text widget above the Table.</p>	
<p>Click on the second text widget and open the <i>Styling</i> panel. There, enter “Title_Complaints” as the Name.</p>	

<p>To edit what appears in the box, double click on it in the Canvas and enter "Complaints" in the text widget above the Table.</p>	
<p>Now, we will add the R Visualization widgets. To do that, select the widget from the <i>Insert</i> panel and insert 2 into the Canvas and place them vertically next to the Chart.</p>	
<p>Select the first R Visualization widget in the Canvas and open the Designer to edit its properties. We will start in the <i>Styling</i> panel. There, enter "RVisualization_WordCloud_2018" as the Name.</p>	
<p>To edit its content, let's switch over to the <i>Builder</i> panel. Here, enter the data set as the input data and check the "Refresh On Resize" option.</p>	

<p>After inserting the data source (here: BestRunBike_Customer_Complaint), click on it (still in the <i>Builder</i> panel) so that we can edit the properties that we want the data set to have.</p> <p>Here, we will add Complaint Category to the Rows and Account to the Columns.</p> <p>Click on Add Filters and select Date – Range and choose Year 2018 to 2018.</p> <p>The filters should now be Category set to Actual and Date set to 2018-2018.</p>	
<p>Click on OK and back in the <i>Builder</i> panel of the widget, click on Edit Script.</p>	

In the R script of this widget, we will get the words from the complaints and also how frequent they come up and according to these values, the word cloud is generated and the words with the higher frequency are also drawn bigger in the word cloud.

Insert the script written on the right side, into the editor of the R widget and click on Apply to save the changes.



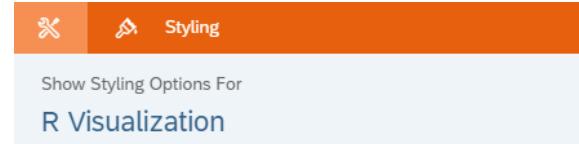
```

# load package
library(wordcloud)

# get words
words <- BestRunBike_Customer_Complaint$`Complaint Category`
# get frequency
frequency <- BestRunBike_Customer_Complaint$Count
if (exists("colorValue")) {
  myColor <- colorValue
} else {
  myColor <- "Oranges"
}
# generate word cloud
wordcloud(words, frequency, scale = c(4, 1),
          rot.per=0.2, colors=brewer.pal(8, myColor))

```

Now, let's do the same for the second R Visualization widget. Select it in the Canvas and open the *Styling* panel. There, enter "RVisualization_WordCloud_2019" as the Name.

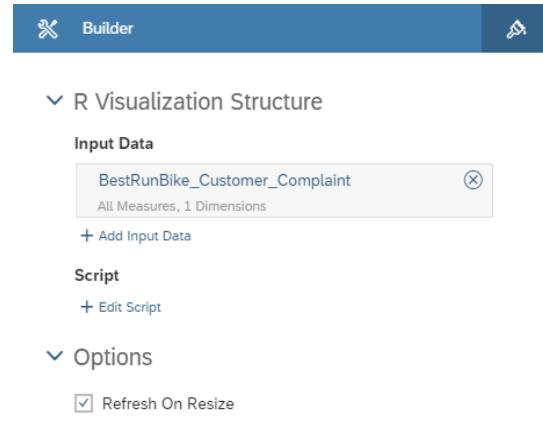


Application Design Properties

Name

RVisualization_WordCloud_2019

To edit its content, let's switch over to the *Builder* panel. Here, enter the data set as the input data and check the "Refresh On Resize" option.

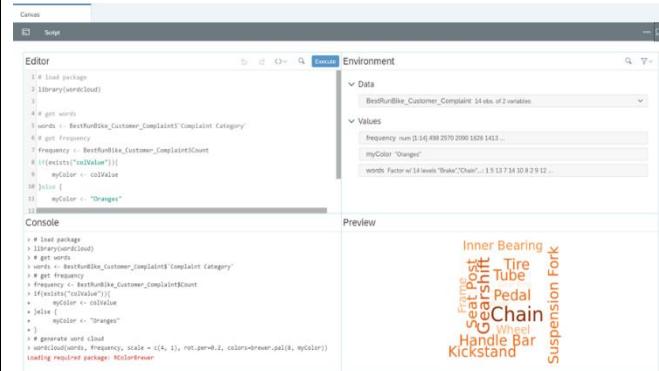


<p>After inserting the data source, click on it (still in the <i>Builder</i> panel) so that we can edit the properties that we want the data set to have.</p> <p>(We will have the same settings that we had for the last widget but change the date to 2019.)</p> <p>Here, we will add Complaint Category to the Row and in the Columns, we will add Account.</p> <p>Click on Add Filters and select Date – Range and choose Year 2019 to 2019.</p> <p>The filters will be Category set to Actual and Date set to 2019-2019.</p>	
<p>Click on OK and back in the <i>Builder</i> panel of the widget, click on Edit Script.</p>	

In the R script of this widget, we will do the same as in the first widget; we will get the words from the complaints and how frequent they come up and according to these values, the word cloud is generated and the words with the higher frequency are also drawn bigger in the word cloud.

Insert the script written on the right side, into the editor of the R widget.

Click on Apply to save the changes.



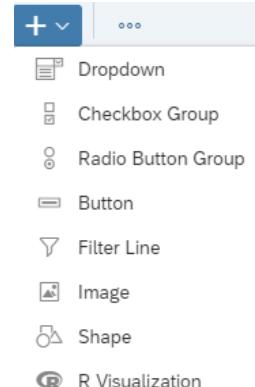
```
# load package
library(wordcloud)

# get words
words <- BestRunBike_Customer_Complaint$`Complaint Category`

# get frequency
frequency <- BestRunBike_Customer_Complaint$Count
if (exists("colValue")) {
  myColor <- colValue
} else {
  myColor <- "Oranges"
}

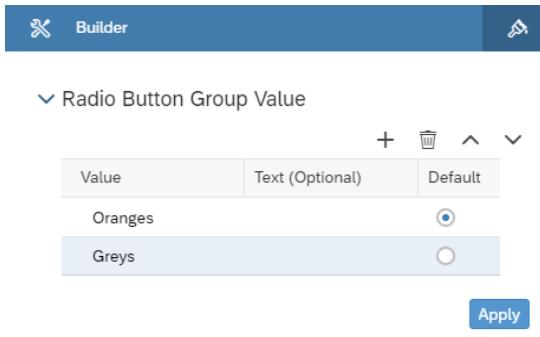
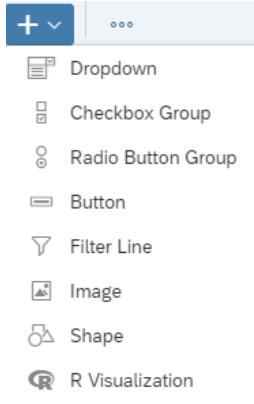
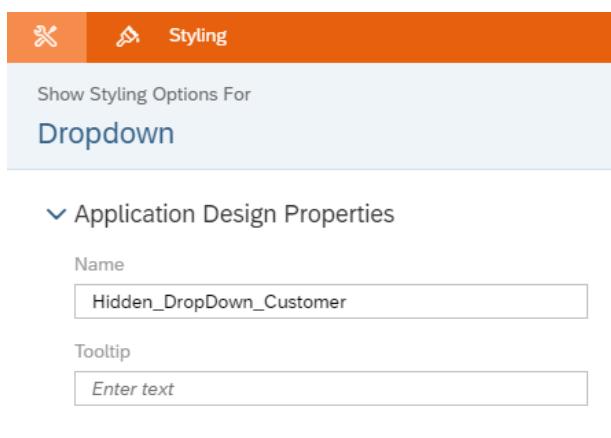
# generate word cloud
wordcloud(words, frequency, scale = c(4, 1),
          rot.per=0.2, colors=brewer.pal(8, myColor))
```

To give the users more choice in the look of the R Visualization widgets, we want them to be able to choose the color of the generated word cloud. To do this, we need to have a Radio Button Group where we will give them the choice between 2 colors.
Add a Radio Button Group widget and place it above the R Visualization widgets.



Select the widget in the Canvas and open the *Styling* panel. Here, we will edit its Name and set it to “RadioButtonGroup_Color” and set the Display Option to Horizontal Layout.

Name	<input type="text" value="RadioButtonGroup_Color"/>
Display Option	<input type="button" value="Horizontal Layout"/>

<p>To edit the content of the Radio Button Group widget, switch over to the <i>Builder</i> panel. There, we will add 2 values “Oranges” and “Greys”, while setting Oranges to our default.</p> <p>To save the changes, click on Apply.</p>	
<p>The last widget we will need for this application is a Dropdown list. To add a Dropdown list, go to the <i>Insert</i> panel and select a Dropdown widget.</p>	
<p>Firstly, we need to change the name of the widget to make it more comprehensible if we want to call it in a script. To do that, select the widget in the Canvas and go to the <i>Styling</i> panel. There, enter “Hidden_DropDown_Customer” in the Name field.</p>	

Through the `getSelection` function of the Table, we get back a dimension ID and a member ID when a user selects an element in the Table. This ID is very useful and allows us to manipulate widgets, however, if we want to be able to display the name of the selected member (here: the name of the selected customer), we have to get the text of the name using the member ID we get from the Table's function.

That's why we need this Dropdown list here, we will simply load all the customers in our data set into it and set its selected key to the captured `memberId`. This way we can get the text of the element and use it to make our Canvas more dynamic.

Thus, we don't need this widget to be visible since we just need it for behind-the-scenes work.

To set the widget to invisible, hover over



it in the Layout and click on the icon.

Once there, click on Hide to make the widget invisible in the Canvas.

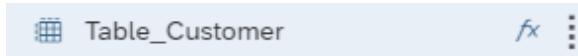


To enable the user to filter the Chart and the R Visualization widgets according to a specific customer, we will write a script for the Table so that when a user selects a specific customer from our Top 10 Customers list, all our other widgets are filtered to that specific customer.

To access the script of the Table, hover over the widget in the Layout and click



on the icon next to it and select `onSelect`.



In this `onSelect` event script, we will capture the selected element and get its dimension (here: it's already known that it's Customer) and the selected member ID (the specific customer selected). We will then use these values to add filters to the Chart, and the two R Visualization widgets.

At the end of the script, we will use our Hidden Dropdown list to get the Text related to the memberId we get back from the `getSelection` function and edit the text of the Complaint Chart's label to include the name of the selected customer.

```
Canvas Table_Customer - onSelect ×
Table_Customer - onSelect
Called when the user makes a selection within the table.
Function onSelect(): void

1 var sel = Table_Customer.getSelections();
2
3 console.log(["Sel ", sel]);
4
5 if (sel.length > 0){
6   var selection = sel[0];
7   console.log(['Selection [0] : ', selection]);
8
9   for (var dimensionId in selection){
10     var memberId = selection[dimensionId];
11     console.log(['Selection Dimension: ', dimensionId]);
12     console.log(['Selection Member: ', memberId]);
13     Chart_Complaints.getDataSource().setDimensionFilter(dimensionId, memberId);
14     RVisualization_WordCloud_2018.getDataSource("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter(dimensionId, memberId);
15     RVisualization_WordCloud_2019.getDataSource("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter(dimensionId, memberId);
16     Hidden_DropDown_Customer.setSelectedKey(memberId);
17     var text = Hidden_DropDown_Customer.getSelectedText();
18     Title_Complaints.applyText("Complaints for Customer " + text);
19   }
20 }
```

```
var sel = Table_Customer.getSelections();

console.log(["Sel ", sel]);

if (sel.length > 0) {
  var selection = sel[0];
  console.log(['Selection [0] : ', selection]);

  for (var dimensionId in selection) {
    var memberId = selection[dimensionId];
    console.log(['Selection Dimension: ', dimensionId]);
    console.log(['Selection Member: ', memberId]);

    Chart_Complaints.getDataSource().setDimensionFilter(dimensionId, memberId);

    RVisualization_WordCloud_2018.getDataSource("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter(dimensionId, memberId);

    RVisualization_WordCloud_2019.getDataSource("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter(dimensionId, memberId);
    Hidden_DropDown_Customer.setSelectedKey(memberId);
    var text =
    Hidden_DropDown_Customer.getSelectedText();
    Title_Complaints.applyText("Complaints for Customer " + text);
  }
}
```

The next script we will write is for the Region Radio Button Group. To access the script, hover over the widget in the Layout and click on the  icon next to it.

 RadioButtonGroup_Region ... 

In this script, we will add countries to the Radio Button Group of Country according to what region is selected. For example, if Region 2 (EMEA) is selected, then Dubai, Germany, and Great Britain are displayed as options in the Countries widget, however, if the region changes to another, for example, Region 4 is chosen (APJ), then the countries that the user can choose from in the other Radio Button Group are India, China, and Australia.

Furthermore, we will set the dimension filter of the widgets in our Canvas according to the selected region.

```

Canvas <-- RadioButtonGroup_Region -> onSelect
Called when the user changes the selection in the radio button group.
Function onSelect(): void

1 var sel = RadioButtonGroup_Region.getSelectedKey();
2
3 RadioButtonGroup_Country.removeAllItems();
4 RadioButtonGroup_Country.addItem("ALL", "All");
5 RadioButtonGroup_Country.setSelectedKey("ALL");
6
7 if (sel === "REGION01") {
8   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY011]", "Mexico");
9 } else if (sel === "REGION02") {
10   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY021]", "Dubai");
11   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY022]", "Germany");
12   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY023]", "Great Britian");
13 } else if (sel === "REGION03") {
14   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY031]", "USA East");
15   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY032]", "USA West");
16   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY033]", "Canada");
17 } else if (sel === "REGION04") {
18   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY041]", "India");
19   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY042]", "China");
20   RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY043]", "Australia");
21 }
22
23
24 Table_Customer.getDataSource().setDimensionFilter("Region", sel);
25 Chart_Complaints.getDataSource().setDimensionFilter("Region", sel);
26 RVISUALIZATION_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Region", sel);
27 RVISUALIZATION_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Region", sel);
28
29 Table_Customer.getDataSource().removeDimensionFilter("Country");
30 Chart_Complaints.getDataSource().removeDimensionFilter("Country");
31 RVISUALIZATION_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country");
32 RVISUALIZATION_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country");

var sel = RadioButtonGroup_Region.getSelectedKey();

RadioButtonGroup_Country.removeAllItems();
RadioButtonGroup_Country.addItem("ALL", "All");
RadioButtonGroup_Country.setSelectedKey("ALL");

if (sel === "REGION01") {

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY011]", "Mexico");
} else if (sel === "REGION02") {

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY021]", "Dubai");

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY022]", "Germany");

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY023]", "Great Britian");
} else if (sel === "REGION03") {

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY031]", "USA East");

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY032]", "USA West");

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY033]", "Canada");
} else if (sel === "REGION04") {

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY041]", "India");

  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY042]", "China");

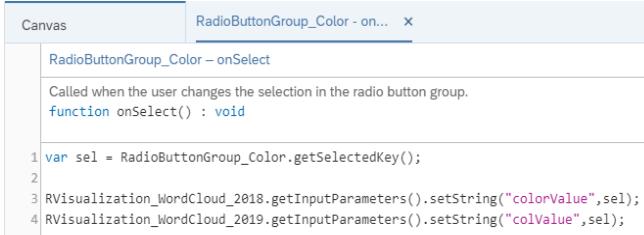
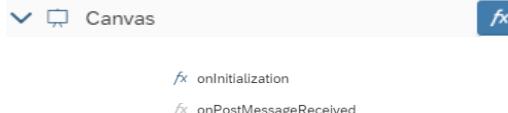
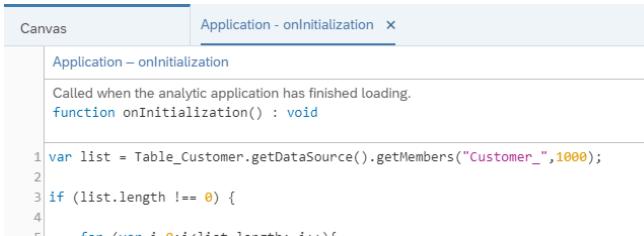
  RadioButtonGroup_Country.addItem("[Country].[Region].&[COUNTRY043]", "Australia");
}

Table_Customer.getDataSource().setDimensionFilter("Region", sel);
Chart_Complaints.getDataSource().setDimensionFilter("Region", sel);
RVISUALIZATION_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Region", sel);
RVISUALIZATION_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Region", sel);

Table_Customer.getDataSource().removeDimensionFilter("Country");
Chart_Complaints.getDataSource().removeDimensionFilter("Country");
RVISUALIZATION_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country");
RVISUALIZATION_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country");

```

	<pre>Chart_Complaints.getDataSource().removeDimensionFilter("Country"); RVisualization_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country"); RVisualization_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country");</pre>
<p>Now, we will edit what happens when one of the options in the Radio Button Group Country is selected.</p> <p>To do that, hover over the widget in the Layout and click on the  icon that appears next to it.</p>	
<p>In this script, we will simply set the selected option as the dimension filter of our Table, Chart, and R Visualization widgets.</p> <p>However, because the R Visualization widget needs a different kind of input than the Table and the Chart, we need to edit the key we get and cut some of it so that we can forward it to the R widgets.</p>	<pre>Canvas RadioButtonGroup_Country - ... RadioButtonGroup_Country - onSelect Called when the user changes the selection in the radio button group. function onSelect() : void 1 var sel = RadioButtonGroup_Country.getSelectedKey(); 2 3 if (sel === "All") { 4 Table_Customer.getDataSource().removeDimensionFilter("Country", sel); 5 Chart_Complaints.getDataSource().removeDimensionFilter("Country", sel); 6 RVisualization_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Region", sel); 7 RVisualization_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Region", sel); 8 } else { 9 Table_Customer.getDataSource().removeDimensionFilter("Country"); 10 Chart_Complaints.getDataSource().removeDimensionFilter("Country"); 11 RVisualization_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country"); 12 RVisualization_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country"); 13 } var sel = RadioButtonGroup_Country.getSelectedKey(); var cloud_sel = sel.replace("[Country].[Region].&[", ""); cloud_sel = cloud_sel.replace("[", " "); console.log(cloud_sel); if (sel === "All") { Table_Customer.getDataSource().removeDimensionFilter("Country"); Chart_Complaints.getDataSource().removeDimensionFilter("Country"); RVisualization_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country"); RVisualization_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().removeDimensionFilter("Country"); } else { Table_Customer.getDataSource().setDimensionFilter("Country", sel); Chart_Complaints.getDataSource().setDimensionFilter("Country", sel); RVisualization_WordCloud_2018.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Country", cloud_sel); RVisualization_WordCloud_2019.getDataFrame("BestRunBike_Customer_Complaint").getDataSource().setDimensionFilter("Country", cloud_sel); }</pre>

<p>There is now another widget that we have to write the function for; the Color Radio Button Group that controls whether the word cloud will be displayed in Orange or in Gray. To edit the script of this widget, hover over it in the Layout and click on the  icon that appears next to it.</p>	
<p>In the script of this widget, we will simply get the selected option, save it in a variable and pass it as input parameters to the R Visualization widgets' scripts.</p>	 <pre> var sel = RadioButtonGroup_Color.getSelectedKey(); RVisualization_WordCloud_2018.getInputParameters().setString("colorValue", sel); RVisualization_WordCloud_2019.getInputParameters().setString("colValue", sel); </pre>
<p>The last script for this application is the one that gets executed when the application is initialized. To access this script, hover over the “Canvas” in the  Layout, click on the  icon that appears next to it, and select <i>onInitialization</i>.</p>	
<p>In this script, we will load a maximum of 1000 customers into the Hidden Dropdown of Customers. This number was chosen because the number of customers in our data set is under 1000, however, this number can be changed if needed.</p>	 <pre> var list = Table_Customer.getDataSource().getMembers("Customer_", 1000); if (list.length !== 0) { for (var i=0;i<list.length; i++){ console.log(['List Dimension: ', i , list[i].displayId]); console.log(['List Description: ', i , list[i].description]); console.log(['List Member: ', i , list[i].id]); Hidden_DropDown_Customer.addItem(list[i].id, list[i].description); } } var list = Table_Customer.getDataSource().getMembers("Customer_", 1000); if (list.length !== 0) { for (var i = 0; i < list.length; i++) { console.log(['List Dimension: ', i , list[i].displayId]); console.log(['List Description: ', i , list[i].description]); console.log(['List Member: ', i , list[i].id]); Hidden_DropDown_Customer.addItem(list[i].id, list[i].description); } } </pre>

<p>Now let's save the application and see how it looks like.</p> <p>Click on Run Analytic Application in the upper right side of the page and the result should look something like this:</p> <p>If we click on one of the elements in the Table (one of the Customers), the Chart and the R Visualization widgets will be filtered according to the data-related to this particular customer (here: Northside Bikes was selected).</p> <p>Now, if we select Greys instead of Oranges in the Color Radio Button Group, the R Visualization widgets are displayed in gray.</p> <p>To filter according to a certain region, we can select one of the Regions from the Region Radio Button Group (here: EMEA was selected), and consequently the Country Radio Button Group's options changed from (All and Mexico) to EMEA countries (All, Dubai, Germany, and Great Britain). The screenshot on the right displays the widgets filtered on EMEA and Germany; the customer is Greenhigh Bikes.</p>	<p>The screenshot shows a SAP Analytics Cloud dashboard. On the left, there is a table titled "Top 10 Customer" with columns "Customer" and "Count". The data includes: Northside Bikes (1,356), Gearshift Inc. (1,240), Greenhigh Bikes (1,195), The Cycle Factor (1,108), Decathlon US/TM Classroom (1,072), VeloCity Cycles (1,074), Greenhigh Bikes (1,074), Greater Bedford REGION (1,074), SuperGizmos (1,073), and Jason Wright Inc. (1,070). To the right of the table are two R visualization widgets. The top one is a bar chart titled "Complaints" showing counts for various parts like Chain, Pedal, Handle Bar, etc., for the years 2018 and 2019. The bottom one is a word cloud where the size of words represents their frequency. Both charts and the word cloud are color-coded by category: Oranges (red/orange) and Greys (gray). A legend at the top indicates "Oranges" (red circle) and "Greys" (gray circle). The word cloud on the right is specifically filtered for the customer "Greenhigh Bikes".</p>
--	--

6.10 Setting User Input for Planning Data

The user can set values to cells of a planning-enabled tabled using an analytics designer script. After setting one or more specific cell values the user can refresh the Table by submitting the values, for example:

```
Table_1.getPlanning().setUserInput({"sap.epm:Account":  
    "[sap.epm:Account].[parentId].&[TAXES]", "sap.epm:ProfitAndLoss_Version02":  
    "public.Actual"}, "123456789");  
  
Table_1.getPlanning().submitData();
```

The passed value is always an unscaled value (raw value). For example, if the table applies a scaling factor of one million when displaying its cell values, then the value set above is displayed as 123.46 (formatted value). Note the rounding of the last displayed digit of the formatted value.

If the passed value is prefixed with an asterisk (*), then the value is applied as a factor to the present cell value. For example, applying the following script after the script above

```
Table_1.getPlanning().setUserInput({"sap.epm:Account":  
    "[sap.epm:Account].[parentId].&[TAXES]", "sap.epm:ProfitAndLoss_Version02":  
    "public.Actual"}, "*0.5");  
  
Table_1.getPlanning().submitData();
```

results in a cell value (raw value) of 61728394.5, which is displayed as 61.73 (formatted value).

Another example shows a combination of a table with an input field. The value of the input field is applied as the new cell value to the first selected cell of the table:

```
var selectedCell = Table_1.getSelections()[0];
```

```
var planning = Table_1.getPlanning();
planning.setUserInput(selectedCell, InputField_1.getValue());
planning.submitData();
```

Currently, the passed value (raw value) can have up to 17 characters if it's a new value and up to 6 characters if it's a factor.

7 Planning

7.1 What to Expect from Analytics Designer Regarding Planning?

Analytics designer reuses the Planning features of SAP Analytics Cloud and leverage the capabilities by offering flexible scripting that supports customizations of applications according to user requirements. Planning Data Models, Allocations, Data Action Triggers, and all Planning features can be integrated to applications.

And what can you not expect? In analytics designer you can't use Input Tasks and Planning scripting isn't possible for models based on BPC Write-Back.

7.2 Basic Planning Concepts in Analytics Designer

Planning specific features can be triggered through the toolbar icons in the *Plan* area.



Figure 73: Toolbar Planning Features

These icons are greyed out if no table cell with a planning model is selected.

Most of these features can also be triggered through scripting.

To get the Planning Table object, use the script below. If the table has no planning model assigned, it will return `undefined`.

```
Table.getPlanning(): Planning | undefined
```

Scripting will perform the same planning actions that could be done via UI. The benefits of scripting are augmented in cases which you want to minimize the number of clicks from your user, personalize your UI or when a special customer requirement can't be fulfilled with standard planning behavior.

Data can't be changed during design time, and you can enable the usage of planning features during runtime in two different ways:

- In the table designer UI: You can find in the *Builder* panel, section *Properties*, a box called *Planning enabled*.

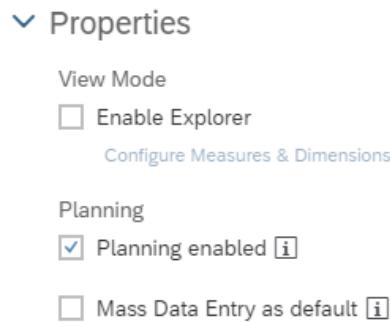


Figure 74: Planning Enabled

- Through the script below:

```
setEnabled(boolean): void
```

This option can be useful when you shall disable planning due to specific requirements. For example, budget might not be changed in last quarter of the year.

One other valuable script allows checking whether the data model is planning-enabled:

```
isEnabled(): boolean
```

In the Table's *Builder* panel, there are some configurations that you can do in each dimension, and **Unbooked Data** might be a good choice when, for example, your Planning Data Model has no booked data, and your end users need to see which dimension members are available for planning.

The screenshot shows the 'Builder' panel for a table. The 'Data Source' is set to 'LivingCompany'. In the 'Table Structure' section, under 'Rows', the 'G/L Account' dimension is selected. A context menu is open over the 'G/L Account' dimension, with the 'Unbooked Data' option highlighted. Other options in the menu include 'Thresholds', 'Add Calculation...', 'Edit Calculations...', 'Add Measure Input Control...', 'Hierarchy...', 'Display Options', and 'Properties'.

Figure 75: Unbooked Data

7.3 Refreshing Your Data

This feature isn't exclusive to Planning and affects all data models and widgets of your application. It can be reached in two different ways:

- By clicking on the first icon of the toolbar.



- Through the script below:

```
Application.refreshData(): void
```

Scripting is useful when, for example, you use data models with Live Connectivity and the end user wishes to refresh data because a background process that updates master data has finished after the application was opened.

7.4 Setting User Input

Instead of having to guide a business user by showing which table cell should be planned, the app designer could create a separate input field. This input field has a pre-informed value from a table selection. The changed value is then updated on a planning model.

The picture below represents the scenario mentioned above to explain this feature:

CATEGORY	Actual	Forecast	Simulation_Opt...
	Actuals	Forecast	Forecast
ACCOUNTALLOC			
Income Statement	-4,660.24	26.17	38.67
Taxes	-7.00	-4.00	-4.00
Taxes	-7.00	-4.00	-4.00
General and Administrative Expenses	-4,705.33	-158.59	-146.09
Other Expenses	-483.66	89.00	76.90
Freight to Customer	-483.66	89.00	76.90

Figure 76: SetUserInput

In this example, the following scripting would be included on the *Save Data* button.

To update a cell of a table with the given value:

```
setUserInput(selectedData: Selection, value: String): boolean
```

Few considerations for this script:

- Value can be maximum 17 characters.
- If value is scaled, then it shall be less than 7 digits.
- It can be performed from a widget or from a table event.

Regarding data formatting – it takes as parameter either a raw value in the user formatting setting ("1234.567" with "." grouping separator) or a scale in the user formatting setting (for example, "*0.5" to divide the value by half or "*2" to double the value) – both of type string.

Example: (scaling in million)

- If you plan “12345678” the formattedValue will be “12.35”.
- If you plan “123456789” the formattedValue will be “123.46”.
- If you plan “*0.5” of rawValue “123456789” the rawValue will be “61728394.5” and formattedValue will be “61.73”.

Regarding data validation:

- If an invalid value is planned, error/warning message is returned, and the script API also returns false.
- If the same value is planned twice, the value is set, and the script API returns true.
- If the cell is locked, the script API returns false, and a warning message is shown to the user.

To submit the updated cell data with all the beforehand modified data and to trigger refresh of data:

```
submitData(): boolean
```

7.5 Planning Versions

There are two types of planning versions, private and public.

7.5.1 Private Versions

This data isn't visible to other users and other solutions of SAP Analytics Cloud.

```
getPrivateVersions(): [Array of Planning Private Versions] | empty array  
getPrivateVersion(versionId: String): Planning Private Version | undefined
```

The script below returns the user ID of the user who created this private version.

```
getOwnerID(): String
```

7.5.2 Public Versions

This data is visible to all users and all solutions of SAP Analytics Cloud.

```
getPublicVersions(): [Array of Planning Public Versions] | empty array  
getPublicVersion(versionId: String): Planning Public Version | undefined
```

Both planning version types have IDs.

```
getId(): String
```

You can use it, for example, when calling `getData()`.

```
getDisplayId(): String
```

You can use it, for example, to display the version in dropdowns or texts.

All versions but ‘Actual’ can be deleted.

```
deleteVersion(): boolean
```

7.6 How to Manage Versions

7.6.1 Publishing and Reverting Data Changes

Any change in data in any type of version is automatically saved. This means that even without any active saving action, if the browser is closed by mistake, for example, data will still be there when application is reopened by the same user who changed the data.

But to make this data visible to other users, you can publish the public versions through the following toolbar icon:



Figure 77: Publish Version

After clicking this icon, the dialog below is opened and an action can be taken per model. You can also revert, and all data changes will be discarded.

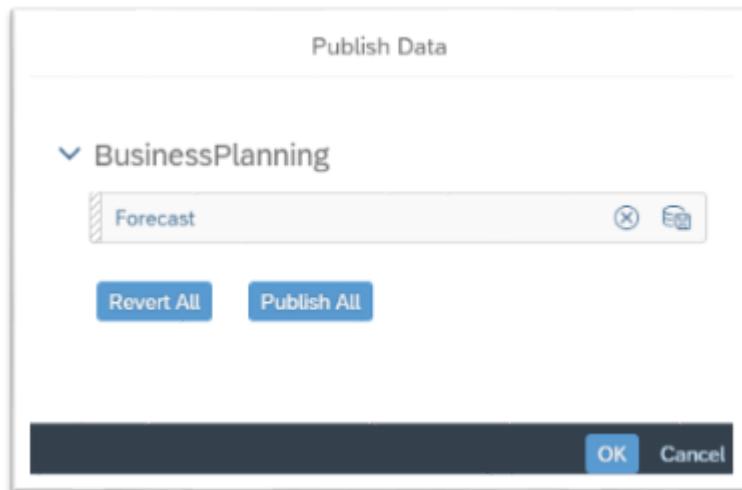


Figure 78: Publish Data

The actions performed within this dialog can also be done via the below scripting on public versions:

```
revert(): boolean  
publish(): boolean
```

After the execution of these scripts, a message informs whether the script ended successfully or not. These are the expected messages:

Version has been published successfully.

Figure 79: Success Message

If the version wasn't modified before these actions are triggered, the message below should be expected:

 You can't publish or revert version 'Forecast' because you have not modified it

Figure 80: Message

This message could be avoided if the dirty check is done in advance.

`isDirty(): boolean`

Dirty versions can be identified by an asterisk (*) just after the version name.

LivingCompany		
VERSION	Actual	Budget*
CATEGORY	Actuals	Budget
G/L ACCOUNT		
> Balance Sheet	-\$16.98 Million	-
> Not Assigned	-\$28,853.15 Million	-
> Net Income	\$806.65 Million	-\$400.00 Million

Figure 81: Dirty Version

It's also possible to publish private versions via the two scripting options below:

`publish(): boolean`

`publishAs(newVersionName: String, versionCategory: PlanningCategory): boolean`

In the second option, a version name is given, and a new public version is created under the informed version category.

These scripts can be very useful if your planning model is placed in a popup, for example. As the toolbar is kept in the background Canvas, users don't need to close the popup to then publish the data. With scripting, you can do it directly in the popup!

VERSION	Actual	Budget*	new budget	Forecast	Strategic Plan
CATEGORY	Actuals	Budget	Budget	Forecast	Planning
G/L ACCOUNT					
22600000	-\$175.19 Million	—	—	—	—
> Balance Sheet	-\$16.98 Million	—	—	—	—
> Not Assigned	-\$28,853.15 Million	—	—	—	—
> Net Income	\$806.65 Million	-\$400.00 Million	-\$564.00 Million	\$1,301.94 Million	\$1,227.67 Million
> Calculated KPIs	\$12,306.75 Million	-\$5,765.89 Million	-\$8,129.90 Million	\$17,865.60 Million	\$17,261.99 Million
Price	—	\$53,400.00	\$53,400.00	\$161,380.00	\$161,380.00
Volume	—	4,395,923.28	4,395,923.28	13,539,443.70	13,707,717.47

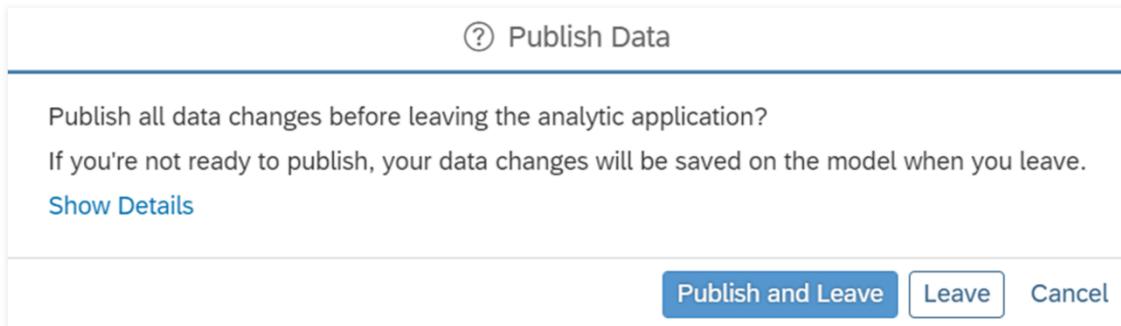
Publish Revert Cancel

Figure 82: Planning Table in Popup

Find in the next section more information about version category and how to create private versions.

Publish and Leave Dialog for Planning

Before leaving the analytic application, check whether to publish your data changes. You'll also be reminded when leaving the application without publishing:



To not show this dialog at runtime, at design time, go to the *Styling* panel of Canvas, and under *Planning Settings* deselect *Remind of publishing all data changes before leaving*.

7.6.2 Copying

Data models with planning-enabled capability have one dimension in common, the version. Each version is classified into one of the following planning categories:

- Actual
- Planning
- Budget

- Forecast
- Rolling Forecast

The version category 'Actual' is created automatically and can't be deleted.

You can use the `PlanningVersion.copy()` script API method to create a private copy of any version:

```
copy(newVersionName: string, planningCopyOption: PlanningCopyOption,  
versionCategory?: PlanningCategory): boolean
```

You can use one of the values of `PlanningCopyOptions` to do the following:

- `PlanningCopyOptions.NoData` – Create a new empty version.
- `PlanningCopyOptions.AllData` – Copy all data from the source version.
- `PlanningCopyOptions.PlanningArea` – Copy only the planning area data from the source version.

7.7 Data Locking

You can use the Data Locking script API to find out if a model is data locking enabled and to set or get the data locking state, even if the table isn't planning-enabled.

The Data Locking script API consists of the following methods:

- `Table.getPlanning().getDataLocking()`
- `Table.getPlanning().getDataLocking.getState()`
- `Table.getPlanning().getDataLocking.setState()`

7.7.1 Using `getDataLocking()`

You can use `getDataLocking()` to check if a model is data locking enabled.

This check is necessary because a user can't perform certain operations on a table, like `setState()` and `getState()`, if the model isn't data locking enabled.

In the following example, the data locking object is retrieved and printed to the console. A data locking object is returned if data locking is enabled on the model.

```
var planning = Table_1.getPlanning();  
console.log(planning.getDataLocking());
```

Note that you can also check if a model is data locking enabled in SAP Analytic Cloud by checking the model preferences (see figure below).

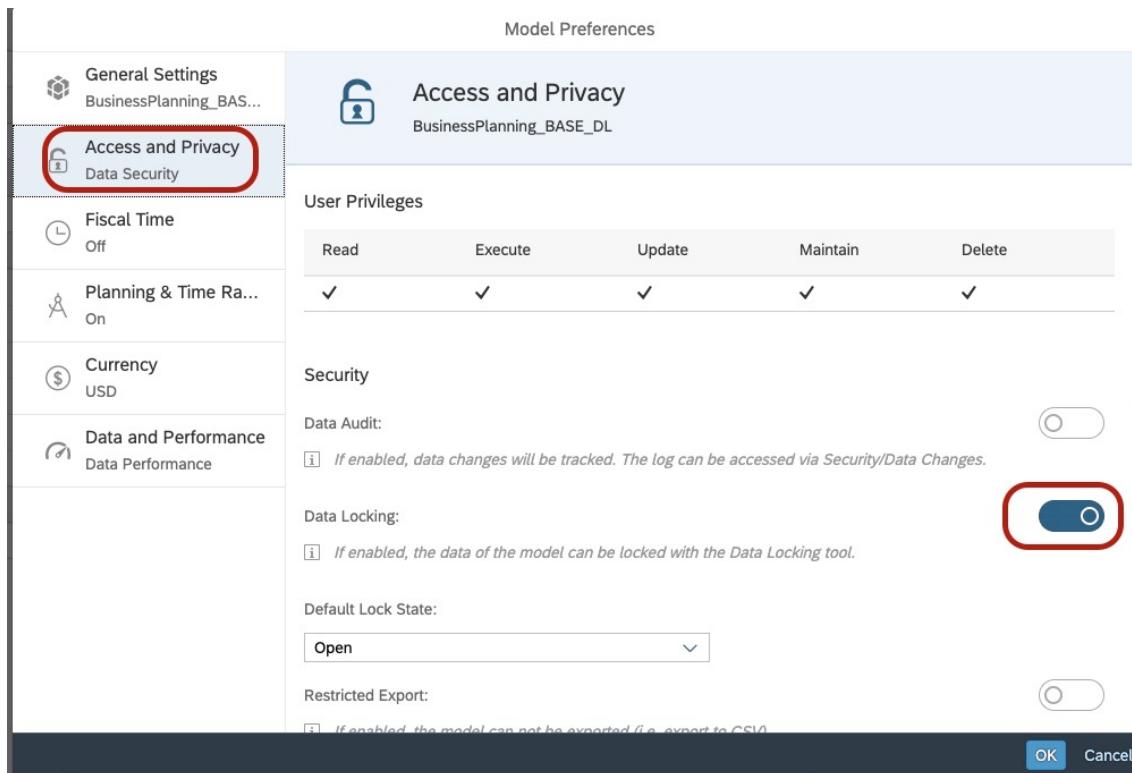


Figure 83: Enabling Data Locking in the Model Preferences

7.7.2 Using `getState()`

You can use `getState()` to get the data locking state of a cell belonging to a driving dimension. This method works only for SAP Analytics Cloud planning models that are data locking enabled.

In following example, the data locking state for a selected cell of a table is retrieved:

```
var selection = Table_1.getSelections()[0];
var selectionLockState =
Table_1.getPlanning().getDataLocking().getState(selection);
```

In order to create a selection on the table, you can either select the cell in the table manually or you can create the selection string yourself in the script editor.

This method returns one of the following values:

- `DataLockingState.Open`
- `DataLockingState.Restricted`
- `DataLockingState.Locked`
- `DataLockingState.Mixed`

If the state of the selection can't be determined, then the method returns `undefined`. This happens if one of the following situations applies:

- The selection is invalid.
- The cell referenced by the selection isn't found.
- The cell is in an unknown state.
- The cell has been created using "Add Calculation" at runtime.

If you've activated the *Show Locks* option for the table, then the "lock" icons will be updated after the method has finished running.

7.7.3 Using `setState()`

You can use `setState()` to set the data locking state of a cell belonging to a driving dimension. This method works only for SAP Analytics Cloud planning models that are data locking enabled.

The method returns `true` if the set operation was successful and `false` otherwise.

You can't set the data locking state on a private version. In this case, the following message is displayed:

"You can only set data locks on public versions. Please use a public version and try again."

You can set one of the following data locking states:

- `DataLockingState.Open`
- `DataLockingState.Restricted`
- `DataLockingState.Locked`

If you attempt to set the data locking state `DataLockingState.Mixed`, then the following message is displayed:

"You can't set the state with the value 'mixed'. Please specify either 'open', 'restricted' or 'locked' as value."

The same message is displayed at runtime if you attempt to execute the script and the script fails.

If you select multiple cells and attempt to set the data locking state, the data locking state will be applied to the first selection only.

In the following example, the data locking state is set for a selected table cell:

```
var selection = Table_1.getSelections()[0];
var isSetStateSuccessful =
Table_1.getPlanning().getDataLocking().setState(selection,
DataLockingState.Locked);
```

Note that if data locking is disabled for a model, all locks will be deleted. If it's turned on again later, all members are reset to their default locking state. The same happens if the default locking state or driving dimensions are changed.

7.8 Planning Events

There are two planning-related widgets that provide an event before their action is triggered.

7.8.1 BpcPlanningSequence

onBeforeExecute

```
onBeforeExecute(): boolean
```

Called when the user clicks the BPC planning sequence trigger. If the method returns true or returns no value, then the BPC planning sequence is executed. If the method returns false, then the BPC planning sequence is ignored.

7.8.2 DataActionTrigger

onBeforeExecute

```
onBeforeExecute(): boolean
```

Called when the user clicks the data action trigger. If the method returns true or returns no value, then the data action is executed. If the method returns false, then the data action is ignored.

7.9 Members on the Fly

You can use the PlanningModel script API to add, update, retrieve, and delete members of dimensions of a planning model. You can also update or delete existing members of a planning model, provided you've the appropriate rights.

Example:

In the following example, a new planning member is added to the dimension "LOCATION" of a planning model:

```
PlanningModel_1.createMembers("LOCATION", {  
    id: "BERLIN",  
    description: "Berlin"  
});
```

Note: You can create multiple members with `createMembers()`. When two members have the same member ID then this results in an error.

Example:

In the following example, the new planning member is updated by adding a data locking owner:

```
PlanningModel_1.updateMembers("LOCATION", {  
    id: "BERLIN",  
    dataLockingOwners: [{  
        id: "ADMIN",  
        type: UserType.User  
    }]  
});
```

In the following example, the description of the new planning member is printed to the browser console:

```
var member = PlanningModel_1.getMember("LOCATION", "BERLIN");  
console.log(member.description);
```

In the following example, the fifth up to the twelfth member of dimension "LOCATION" is returned:

```
var members = PlanningModel_1.getMembers("LOCATION", {  
    offset: "4",  
    limit: "8"  
});
```

The first member has an offset of 0, so the fifth member has an offset of 4. Starting at this offset, the next 8 members are returned.

The PlanningModel script API provides many more features. For more information, see the online API reference.

Note: You can add members to dimensions of type “Generic” only (see the SAP Analytics Cloud modeler). Adding members to dimension of other types, such as, for example, “Account”, “Version”, “Time”, or “Organization” isn’t supported.

Note: If you’ve added, updated, or deleted members, then call `DataSource.refreshData()` or `Application.refreshData()` if you need the chart or table to reflect the modified members in subsequent method calls operating on visible cells or elements of those widgets, for example, `DataSource.getPlanning().getState()`, `DataSource.getPlanning().setState()`, `DataSource.getData()`, or `Planning.setUserInput()`.

Note: After you’ve added members to a very large model (with millions of members) and have refreshed the model with `Application.refreshData()` or `DataSource.refreshData()`, it may happen that not all added members are immediately displayed, for example, in a table associated with the planning model. The same applies to updating members. This is because these operations work asynchronously in the background. Repeat your refresh operation after a short while.

Note: When retrieving planning model members, represented by `PlanningModelMember` objects, the values of specific properties of these members are only returned if the members’ dimension is configured to provide these values. The following table lists the planning model member property and the right/access that must be granted to this member’s dimension in the SAP Analytics Cloud modeler to receive the property’s value:

Planning Model Member Property	Right/Access
<code>dataLockingOwner</code>	Data Locking Ownership
<code>responsible</code>	Responsible
<code>readers</code>	Data Access Control
<code>writers</code>	Data Access Control

Note: When you add your own properties to planning members, use a prefix to avoid name conflicts with existing properties of planning members.

8 Predictive

In analytics designer, there are several predictive features that can help you to explore the data and gain more insights.

8.1 Time Series Forecast

To predict future values of a specific measure for a period of time, you can run Time Series Forecast on historical data in a Time Series Chart. Time Series Forecast can be configured to turn on/off and switch among different algorithms. You could refer to sample *Gain insights into the data* for the usages in detail.

8.1.1 Switching Forecast On and Off

In *Gain insights into the data*, a Time Series Chart, *Chart_Forecast*, is added to show Gross Margin over time. You can turn on Forecast to predict the future trend based on existing historical data.

Basically, Time Series Forecast can be switched on and off via two ways: the entry in context menu at both design time and runtime,

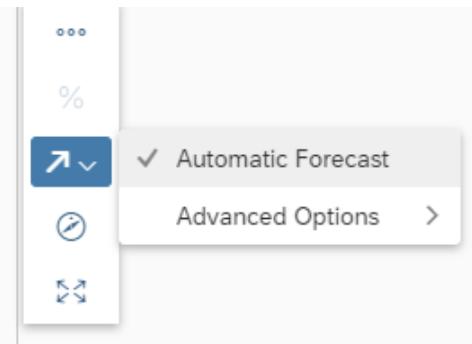


Figure 84: Automatic Forecast

and the script API to set the forecast type:

```
Chart_Forecast.getForecast().setType(ForecastType.Auto);
```

8.1.2 Configuring Forecast

You can also configure the number of periods to predict *Chart_Forecast* for a longer time if needed.

The number of periods to predict can be configured via two ways: the entry in Chart Details at both design time and runtime,

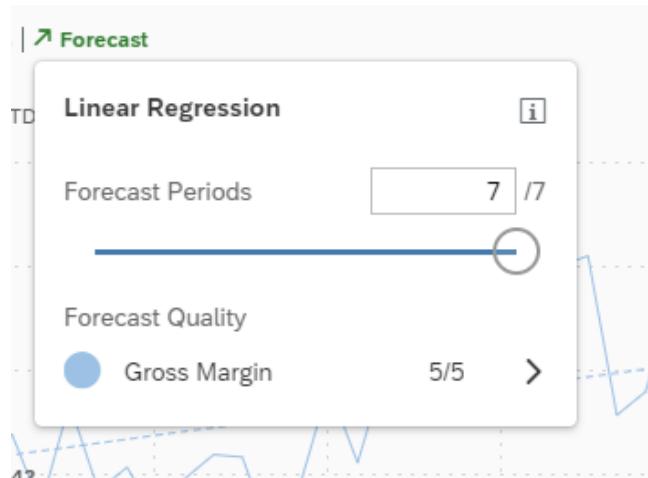


Figure 85: Linear Regression

and the script API to set the value:

```
Chart_Forecast.getForecast().setNumberOfPeriods(7);
```

8.2 Smart Insights

Smart Insights automatically discovers key insights based on existing data. The insights vary among links, correlations, clusters, predictions, and so on. You, as an analytic application developer or as an end user, can straightly investigate the result without any manual exploration. Sample *Gain insights into the data* can be referred to get familiar with the usage.

8.2.1 Discover per Selected Data Point

In *Gain insights into the data*, the Time Series Chart *Chart_Forecast* is added to show Gross Margin over time. You'll notice that the gross margin of May 2015 is low. To get more insights, you can trigger Smart Insights to explore further.



Figure 86: Time Series Chart: Select the Interested Data Point

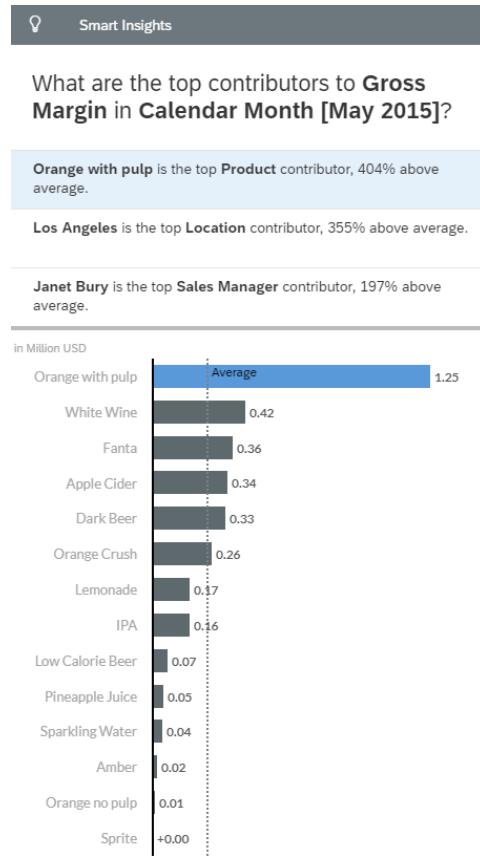


Figure 87: Side Panel of Smart Insights

8.3 Smart Grouping

Smart Grouping can be used to automatically analyze the data points in correlation chart, Bubble or Scatterplot, and group them based on similar properties. You, as an analytic application developer, can configure the visibility of Smart Grouping and the related settings. Sample *Gain insights into the data* demonstrates the typical usages.

8.3.1 Switching Smart Grouping On and Off

In *Gain insights into the data*, a Scatterplot, *Chart_Group*, is added to show Discount and Gross Margin per Store. You can turn on Smart Grouping in this chart to analyze based on similar properties.

Basically, Smart Grouping can be switched on and off via two ways: by the setting in the *Builder* panel at design time,



Figure 88: Smart Grouping

and the script API to set the visibility.

```
Chart_Group.getSmartGrouping().setVisible(true);
```

8.3.2 Configuring Smart Grouping

There are several Smart Grouping settings that you can configure in *Chart_Group*. And you can configure them in two ways: by the entry in the *Builder* panel or Chart Details at both design time and runtime,

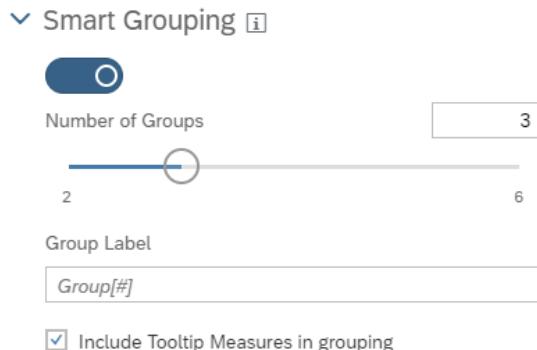


Figure 89: Configure Smart Grouping in *Builder* Panel of Chart

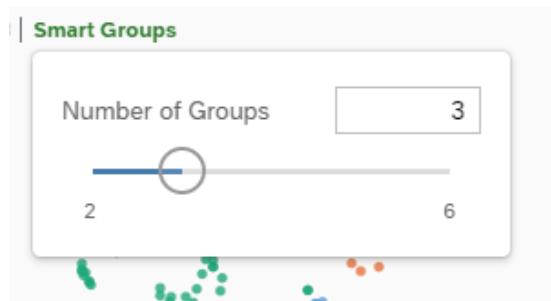


Figure 90: Configure Smart Grouping in Chart Details

and the script API to set the values.

```
Chart_Group.getSmartGrouping().setNumberOfGroups(3);
Chart_Group.getSmartGrouping().setGroupLabel("Group");
Chart_Group.getSmartGrouping().includeTooltipMeasure(true);
```

8.4 Smart Discovery

You, as an analytic application developer, can enable Smart Discovery in your application to discover additional information (for example, key influencers) between columns within a data set.

Sample *Gain insights into the data* demonstrates how to trigger Smart Discovery via the script API.

```
var ds = Chart_Forecast.getDataSource();
var members = ds.getMembers("Product_3e315003an");
var SDsetting = SmartDiscoveryDimensionSettings.create(ds, "Product_3e315003an",
[members[1]]);
SDsetting.setIncludedDimensions(["Location_4nm2e04531", "Store_3z2g5g06m4"]);
```

```

SDsetting.setIncludedMeasures("[[Account_BestRunJ_sold].[parentId].&[Gross_Margin]]"
, "[Account_BestRunJ_sold].[parentId].&[Discount]");
SmartDiscovery.buildStory(SDsetting);

```

In this example, Smart Discovery is invoked via clicking “*More Insights...*” to discover Product with Dark Beer as the target group. In addition, two more measures (Gross Margin and Discount) and two more dimensions (Location and Store) are included in the analysis.

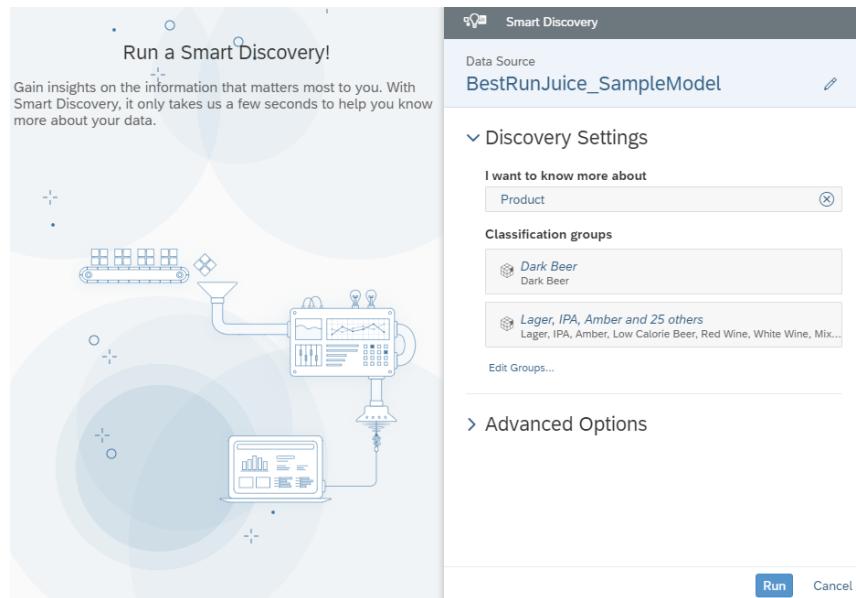


Figure 91: Smart Discovery Setting Panel

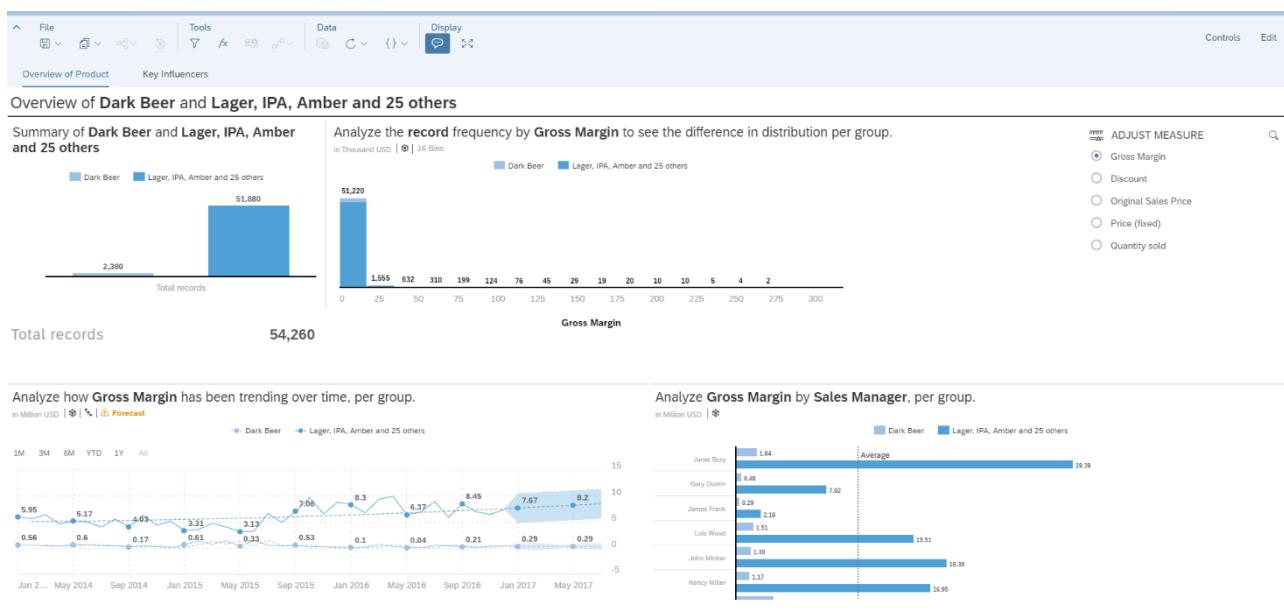


Figure 92: New Document Created by Smart Discovery

8.5 Search To Insight

Search To Insight is a natural language query function that helps users get smart insights on their data.

Create a SearchToInsight Component

To launch a Search To Insight, a SearchToInsight component should be added at design time. You, as an analytic application developer, can configure the data models to search in the side panel of this component.

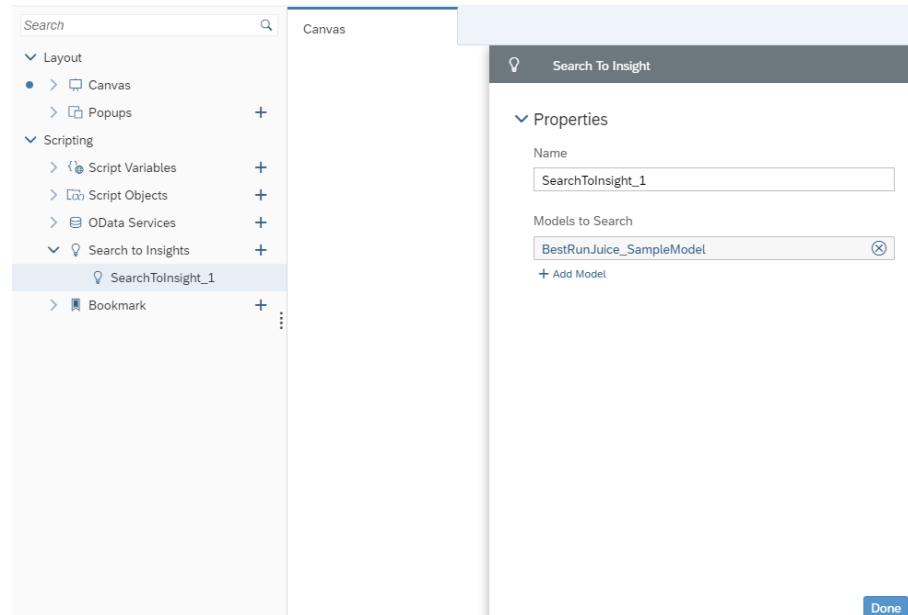


Figure 93: Create a SearchToInsight Component

Launch Search To Insight

Write Analytic Design scripts to launch Search To Insight. At runtime, the analytic application user can open the Search To Insight dialog to get deep and flexible insights of their data.

```
var mode = SearchToInsightDialogMode.Simple;
SearchToInsight_1.openDialog("Gross Margin by Location", mode, true, true);
```

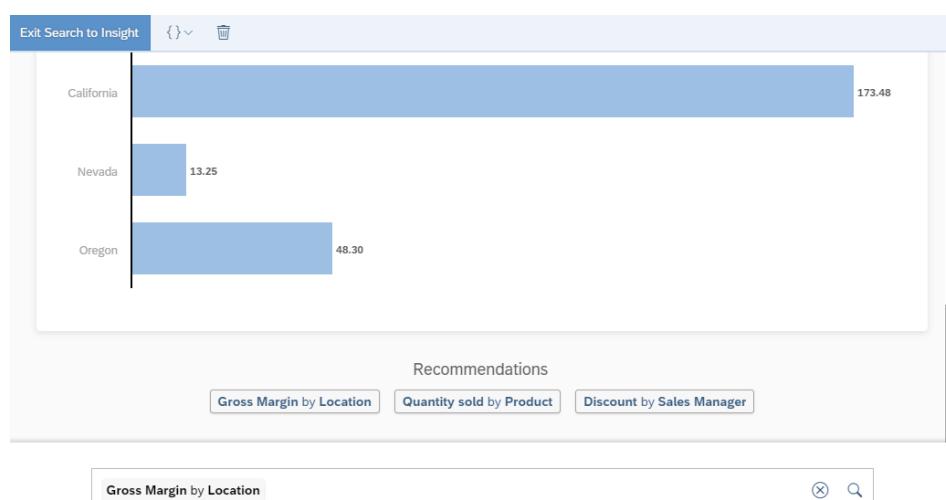


Figure 94: Launch Search To Insight

Apply Search To Insight Results to a Chart

You can also apply Search To Insight results to a chart using the `applySearchToChart()` script API method and leverage the following variable-related script API to save variable value in a Search To Insight component and apply to chart when calling `applySearchToChart()`:

```
// get model variable
SearchToInsight_1.getVariables(modelId: string): VariableInfo[]

// set model variable
setVariableValue(modelId: string, variable: string|VariableInfo, variableValue: string|number|VariableValue|VariableValue[]): void

// search by input question within selected search to insight model and
// apply the results to a chart
SearchToInsight_1.applySearchToChart(question: string, chart: chart): boolean
```

Use Case 1: Trigger Different Modes of Search To Insights

In this example, you can design a simple application to let application users trigger different modes of Search To Insight for the questions they enter.

First, in addition to the scripting object `SearchToInsight_1`, add an input field `InputField_1`, a button `Button_1` and a checkbox group `CheckboxGroup_1` to the Canvas as below:



Figure 95 Search To Insight Example

Then write the following script for the button widget `Button_1`:

```
var mode = SearchToInsightDialogMode.Simple;
if (CheckboxGroup_1.getSelectedKeys().length !== 0) {
    mode = SearchToInsightDialogMode.Advanced;
}
var inputText = InputField_1.getValue();
SearchToInsight_1.openDialog(inputText, mode, false, true);
```

Result: After you save the application and choose *Run Analytic Application*, application users can either trigger simple mode by entering a question and clicking the *Search To Insight* button, or trigger advanced mode by entering a question, selecting *Advanced mode*, then clicking the *Search To Insight* button.

Example 2: Receive Question from Host HTML Page and Apply Search To Insight Results to a Chart

In this example, you can build your own Search To Insight user interface and integrate Search To Insight results to your own portal.

First, embed your analytic application in your own portal via iFrame and maintain the corresponding code to get the message users input in the portal and post it to the embedded analytic application.

Then go back to the analytic application. Besides adding the scripting object `SearchToInsight_1`, write the following script for the `onPostMessageReceived` event of the application:

```
SearchToInsight_1.applySearchToChart(message, Chart_1);
Chart_1.setVisible(true);
```

Result: After that in your own portal, users can ask a question and see the chart of the embedded analytic application appear and display corresponding Search To Insight results.

9 OData

9.1 What You Should Know About OData

The Open Data Protocol (OData) is an open protocol which allows the creation and consumption of queryable and interoperable RESTful APIs in a simple and standard way, initiated by Microsoft in 2007.

Versions 1.0, 2.0, and 3.0 are released under the Microsoft Open Specification Promise.

Version 4.0 was standardized at OASIS, with a release in March 2014. In April 2015 OASIS submitted OData v4 and OData JSON Format v4 to ISO/IEC JTC 1 for approval as an international standard.

“The protocol enables the creation and consumption of REST APIs, which allow Web clients to publish and edit resources, identified using URLs and defined in a data model, using simple HTTP messages. OData shares some similarities with JDBC and with ODBC; like ODBC, OData isn’t limited to relational databases.”

Source: https://en.wikipedia.org/wiki/Open_Data_Protocol

9.2 How You Can Connect to OData

In analytics designer, you can define OData Services based on an existing live connection in your system which was created using CORS (Cross-origin resource sharing) connectivity also referred to as direct connection.

OData Services are supported for the following system types:

- SAP S/4HANA On-Premise
- SAP BW
- SAP HANA
- SAP Business Planning and Consolidation (BPC)

For OData, CORS should be configured on the backend analogous to an InA connection plus support for “if-match” as allowed header.

9.2.1 What You Need to Do

- Define the CORS configuration to your system according to the help page.
- Additionally: Configure support for “if-match” as allowed header in your system.
- Define a direct connection to this system.
- Open an application and add an OData service (more details in the following chapters).

9.2.2 Known Restrictions

In the initial iteration:

- Only parameters of simple types will be supported.
- Actions with mandatory parameters of unsupported types won’t be available.

- For actions with optional parameters of unsupported types, the parameters won't be available but the action itself will.
- In case of bound actions, only binding on entity types (passable by key) will be supported.
- Only the JSON format will be supported.
- Only the following system types are supported:
 - SAP S/4HANA On-Premise
 - SAP BW
 - SAP HANA
 - SAP Business and Consolidation (BPC)
- Only Direct (CORS) connections will be supported. No Path (Proxy), as this feature is being deprecated.

Script execution will block waiting on the response of a triggered action. For now, the assumption is that actions triggering long-running processes return quickly (although the process may not yet be complete). So, while of course the XHR invoking the action is asynchronous, script execution will block waiting for the response, to allow the script writer to react to the return value of the action.

The following types aren't supported:

- Edm.Stream
- Edm.Untyped
- All Edm.Geography types
- All Edm.Geometry types
- All types defined in different namespaces.

9.2.3 What Is an Action?

Actions are operations exposed by an OData service that **may** have side effects when invoked. Actions **may** return data but **must not** be further composed with additional path segments.

9.2.4 What Are Action Imports?

Action Imports or unbound actions aren't associated with any OData EntitySet or Entity. They generally expose simple parameters. All parameters are set via POST body.

9.2.5 What Is a Bound Action?

Bound Actions are actions which may be invoked on a specific Entity. They always contain a first parameter which is set via URL (to resolve the binding to the Entity within the relevant EntitySet), and all other parameters are set via POST body.

In general, actions can be bound on every type, but we support only binding on single entities.

In analytics designer, OData actions can be called from and executed in the backend system via scripting inside an analytic application. Also, programmatic read access to OData services is provided.

9.3 How You Can Call OData Actions

With this feature you, as an analytic application developer, can execute OData (V4) Actions exposed by a connected system within an analytic application.

In your analytic application in the *Layout* section of the *Outline* in the Scripting Section you can create a new OData Service by clicking on plus.

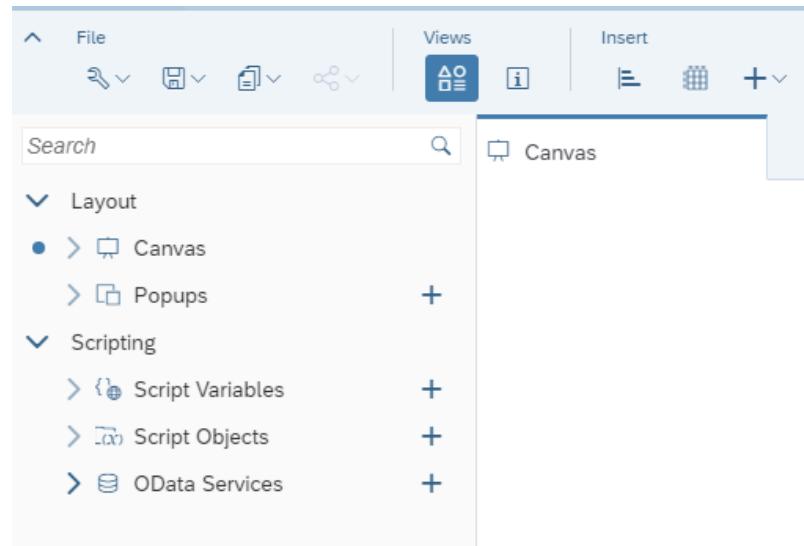


Figure 96: OData Service in *Outline*

Once you've clicked a new entry with the default name ODataService_1 will appear below the node. You'll see a context menu indicated with three points when hovering over the name, where you do the following actions: Rename, Find References, or Delete.

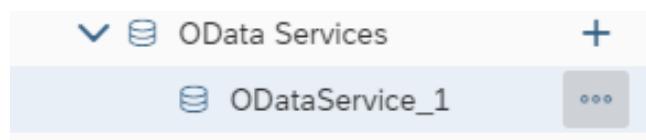


Figure 97: OData Service

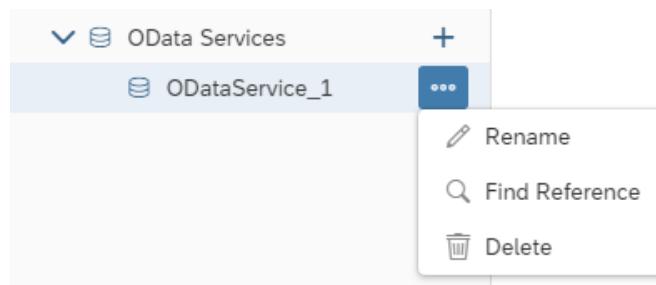


Figure 98: Actions for OData Service in *Outline*

At the same time the side panel opens on the right side. It opens every time you click on the OData Service in the *Outline*. In the side panel you can change the name, select the System from the list of available systems whose connections are already created in SAC under Connections, and specify the End-Point URL of the OData Service manually.

Note: You need to know the URL. So far there is no browse catalog implemented.

To see the metadata of the OData Service you must click the refresh button next to Metadata. Click on Done to close the panel.

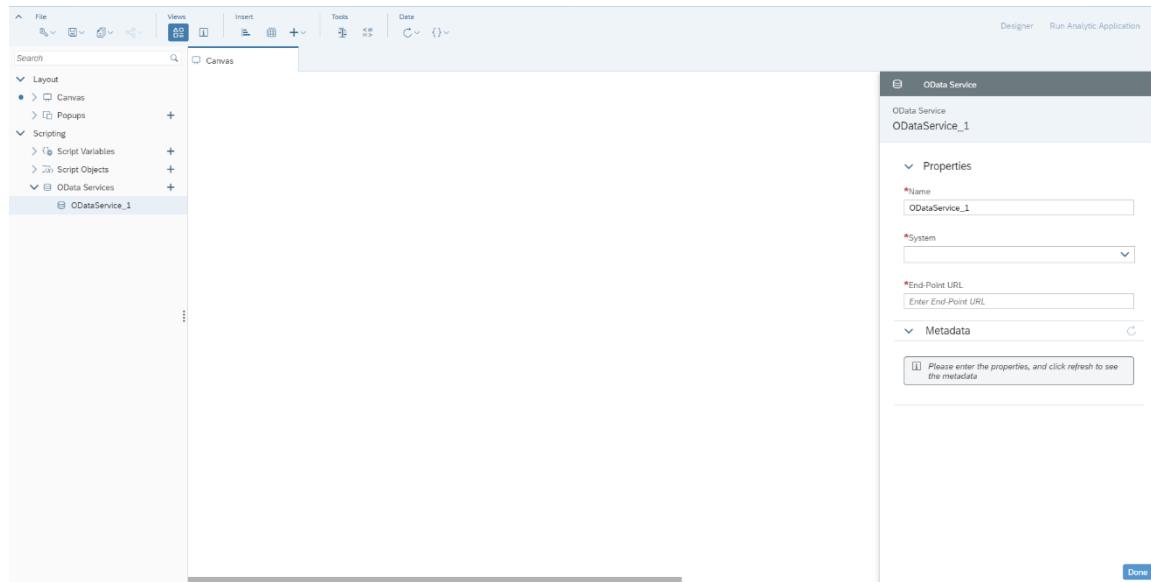


Figure 99: OData Service Side Panel

In the example you see System FUB, the End-Point URL for this OData Service and as Metadata you got the information that this Service is based on OData Version V.4 and it has 2 Actions called Flight/Book and CancelMyFlights.

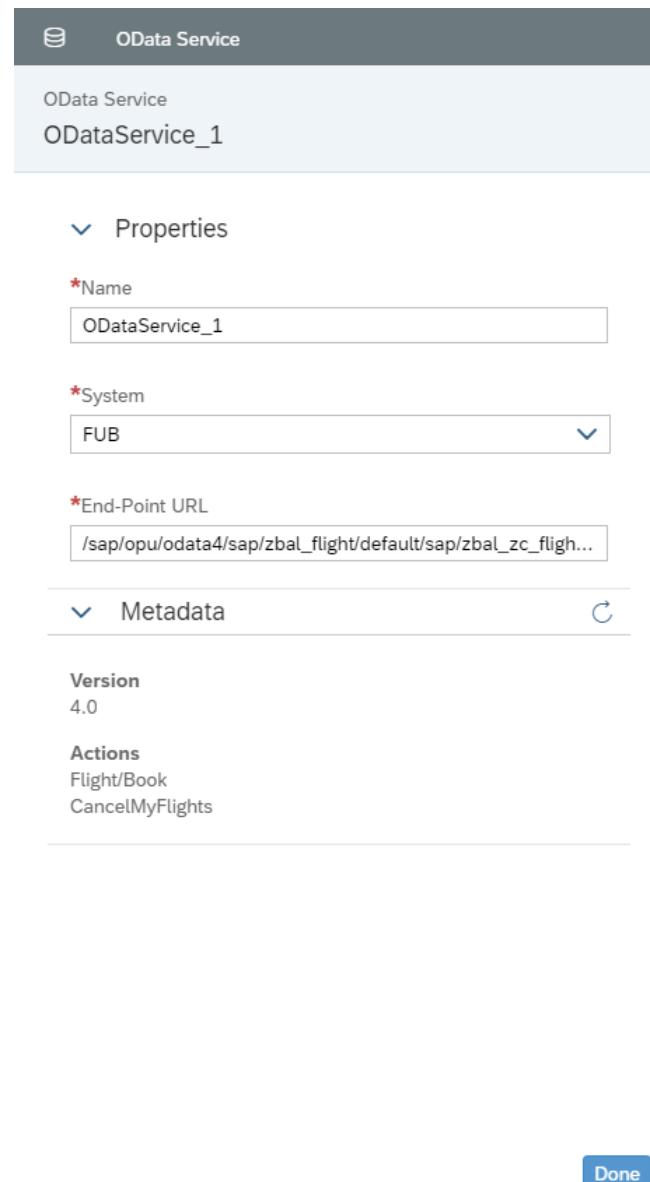


Figure 100: Define OData Service Properties

Now you can insert a Button Widget and change the text of the Button in the *Application Design Properties* of *Styling* panel to Cancel Flight.

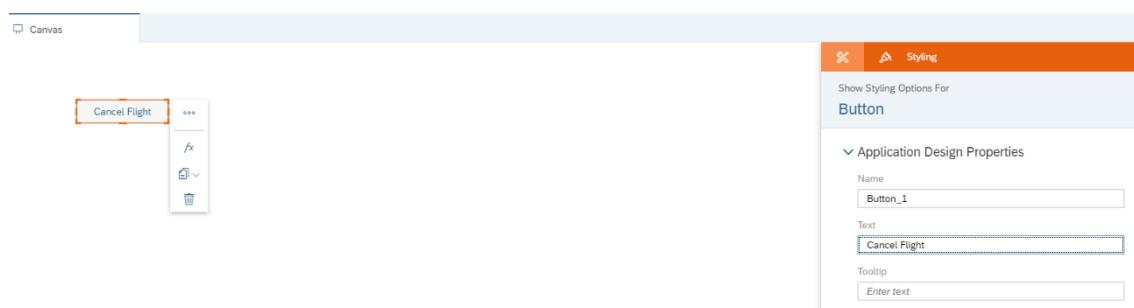


Figure 101: Styling Options

Start the script editor by clicking the  icon in the quick action menu of the widget to create a script which triggers the execution of the action in the source system.

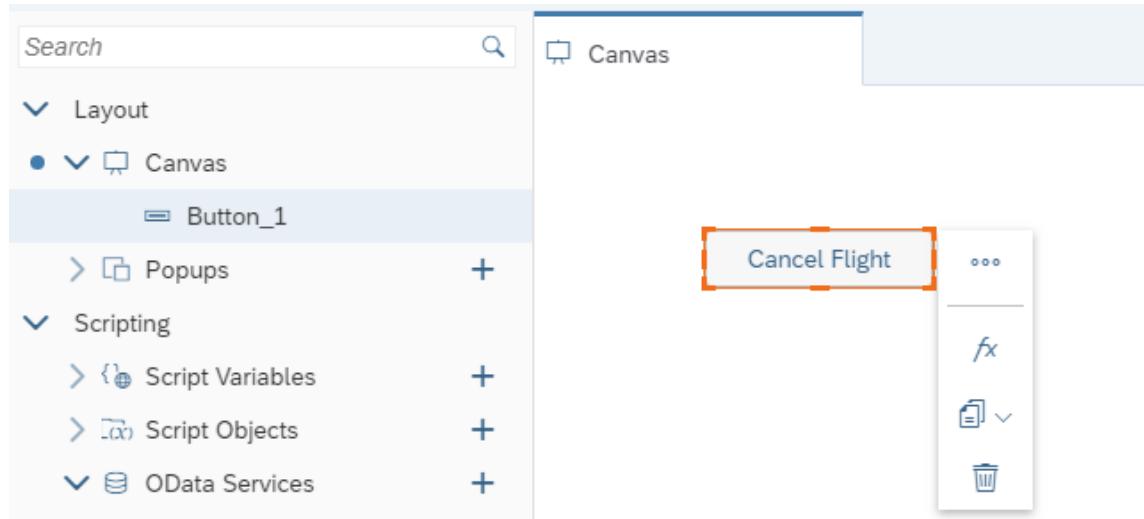


Figure 102: Widget Context Menu

The script editor opens. You can open it as well by hovering over the widget in the *Outline* and clicking the  icon.

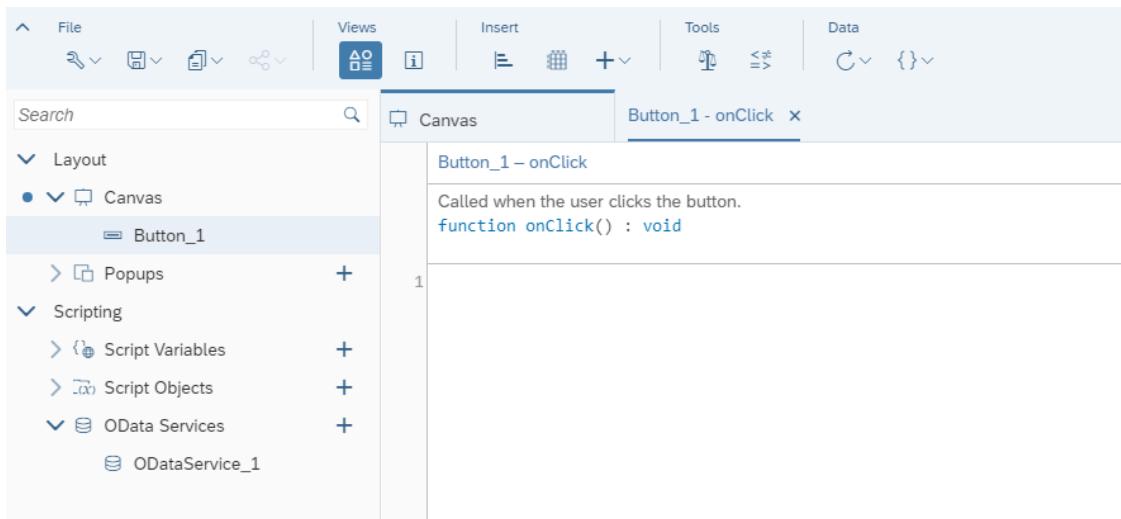


Figure 103: Create Script



Figure 104: Create Script

Type in the name of the OData Service you've specified. The script editor assists you with code completion and value help wherever possible when you click **CTRL+Spacebar**.

The complete expression will look like this:

```
ODataservice_1.executeQuery("CancelMyFlights", {DateFrom: "2019-01-01", DateTo: "2019-12-31"});
```

You've now created the first script to execute an OData action. This Action had a very simple syntax with only 2 parameters.

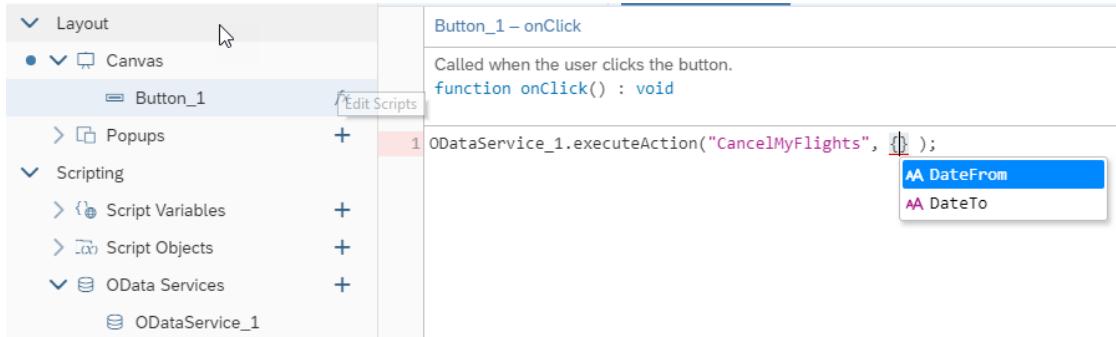


Figure 105: Value Help

Now you can insert another button, rename the text to Book Flight in the *Styling* panel and open the script editor. The BookFlight Action is a bound action which is much more complex than the first one.

The result shall look like this:

```
ODataservice_1.executeQuery("Flight/Book", {Flight: {Airline: "UA", Flightconnection: "0941", Flightdate: "2019-01-05"}, NumberOfSeats: 1});  
ODataservice_1.executeQuery("Flight/Book", {})
```

Figure 106: Value Help for Flight/Book

```
ODataservice_1.executeQuery("Flight/Book", {Flight:{}});
```

Figure 107: Value Help for Flight

Congratulations. You finished the second more complex OData action and now you can run your application and book and cancel a flight for the selected values.

You can enhance your application and start using other script API methods to fill the parameter values dynamically with local or global variables.

Also, you can make the response from the backend system visible in the app by writing the response as message in a text field.

Insert six Text widgets on the Canvas and rename the last one to [MessageBox](#).

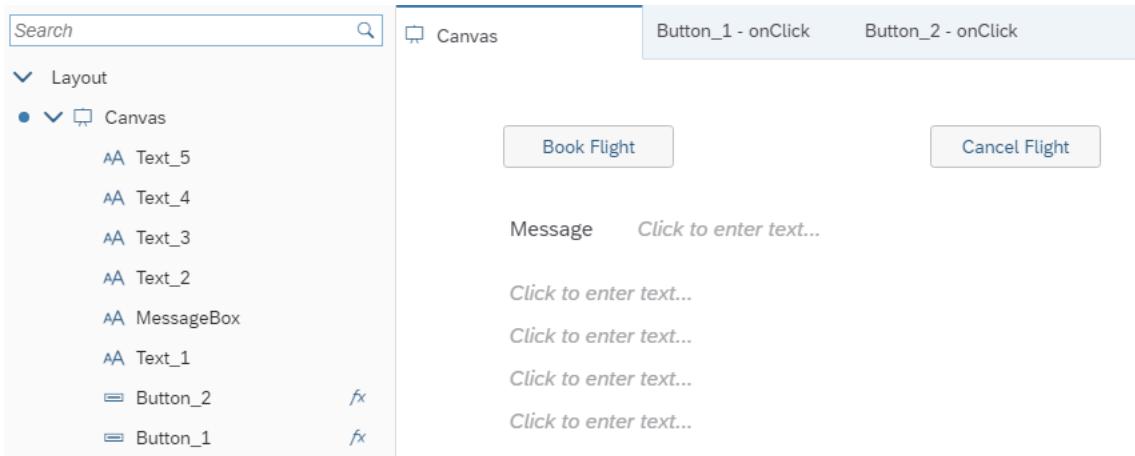


Figure 108: Define Message

Now rewrite your scripts from Book Flight as follows:

```
var ret = ODataService_1.executeAction("Flight/Book",
    {Flight: {Airline: "UA", Flightconnection: "0941", Flightdate:
        "2019-01-05"}, NumberOfSeats: 1});

var succ = "";
if (ret.ok === true) {
    succ = "SUCCESS";
} else {
    succ = "ERROR";
}
MessageBox.applyText(succ);
Text_2.applyText(succ + " message :" + ret.error.message);
Text_3.applyText(succ + " code :" + ret.error.code);
Text_4.applyText("target :" + ret.error.target);
Text_5.applyText("");
```

And rewrite your script from Cancel Flight as follows:

```
var ret = ODataService_1.executeAction("CancelMyFlights", {DateFrom: "2019-01-01",
DateTo: "2019-12-31"});
console.log(ret);
var succ = "";
if (ret.ok === true) {
    succ = "SUCCESS";
} else {
    succ = "ERROR";
}
MessageBox.applyText(succ);
Text_2.applyText(succ + " message :" + ret.error.message);
Text_3.applyText(succ + " code :" + ret.error.code);
Text_4.applyText("target :" + ret.error.target);
var info = "";
if (ret.ok === true) {
    var numberofoccupiedseats =
        ConvertUtils.integerToString(ret.value[0].Numberofoccupiedseats);
    var flightprice = ConvertUtils.numberToString(ret.value[0].Flightprice);
```

```
var totalnumberofseats =
    ConvertUtils.integerToString(ret.value[0].Totalnumberofseats);
var currency = ret.value[0].Currency;
info = "Your flight price was " + flightprice + " " + currency +
    ". " + "There are " + numberofoccupiedseats +
    " occupied from " + totalnumberofseats + " seats in total.";
}
Text_5.applyText("'" + info);
```

Run the application and book a flight and cancel a flight to see the error messages.

To create a meaningful application in the sense of an intelligent application, the best would be to display the backend data via a live connection to a BEx Query. Like this you would be able to see the changes (the booked and canceled seats) in the data directly after clicking the buttons and executing the actions.

9.4 How You Can Read Data from OData Services

Besides OData Actions, there are also many use cases why it makes sense to access EntitySets exposed via OData services.

Therefore, in analytics designer you've programmatic access to these data, which can be used for any purpose beyond the visualization in a table or a chart. For example, you can read and display one member in a text widget.

You can focus on the following capabilities regarding access to OData entity sets:

9.4.1 Retrieving a Single OData Entity

You can retrieve a single OData Entity from an EntitySet, by specifying the key to the entity (analogous to selecting a single row from a SQL table via `SELECT * FROM T1 WHERE ID = <id>`) using the following script API:

```
getEntitiesFromEntitySet(entitySetName: string):
    ODataResult<EntityTypeSpecificPayload[]>
```

where `ODataResult` is the same result structure returned when executing actions. It contains generic information about whether the raw request on HTTP level was successful, and additionally the response payload in success cases:

```
ODataResult<T>: {
    ok: boolean;
    value: T; // depends on payload of action or entity type
    error: ODataError;
}

ODataError: {
    code: string;
    message: string;
    target: string;
    details: ODataError[];
}
```

9.4.2 Retrieving All Entities from an OData EntitySet

You can retrieve all Entities (throttled to a maximum number of 1000) from an OData EntitySet (analogous to `SELECT TOP <N> * FROM T1`).

Example:

```
ODataservice_1.getEntitiesFromEntitySet("Equipments");
```

9.4.3 Retrieving Specific Entities from an OData EntitySet

You can retrieve specific Entities from an OData EntitySet by applying the following OData query options:

- `filter`
- `orderby`
- `select`
- `skip`
- `top`

using the following script API:

```
getEntitiesFromEntitySet(entitySetName: string, options?: ODataQueryOptions):  
    ODataResult<EntityTypeSpecificPayload[]>
```

Example:

In the following example a list of Entities is returned, filtered by the specified query options:

```
ODataservice_1.getEntitiesFromEntitySet("Departments", {filter: "contains(Sector,  
'Consulting')", orderby: "Sector asc"});
```

9.4.4 Retrieving the Number of Entities from an OData EntitySet

You can retrieve the number of Entities from an OData EntitySet by applying the following OData query options:

- `filter`
- `orderby`
- `select`
- `skip`
- `top`

using the following script API:

```
getEntitiesCountFromEntitySet(entitySetName: string, options?: ODataQueryOptions):  
    ODataResult<EntityTypeSpecificPayload[]>
```

Example:

In the following example, the number of entities in the entity set "TEAMS" is returned:

```
ODataservice_2.getEntitiesCountFromEntitySet("TEAMS");
```

In the following example, the number of entities in the entity set "TEAMS" is returned, where the first five entities are skipped and the next ten entries are returned:

```
ODataservice_2.getEntitiesCountFromEntitySet("EMPLOYEES", {skip: 5, top: 10});
```

9.4.5 Known Restrictions

As of today, the following features aren't supported:

- Chaining from one EntitySet to another
- Expand (analogous to joining)
- EntitySets with parameters and EntitySets with mandatory filters

10 Post Message API

When you embed an analytic application in a host HTML page or embed a web page in analytic application through the web page widget, you can follow this guide to enable message communication between host and embedded web pages.

Using the Post Message script API, you as the application developer can realize either of the following scenarios:

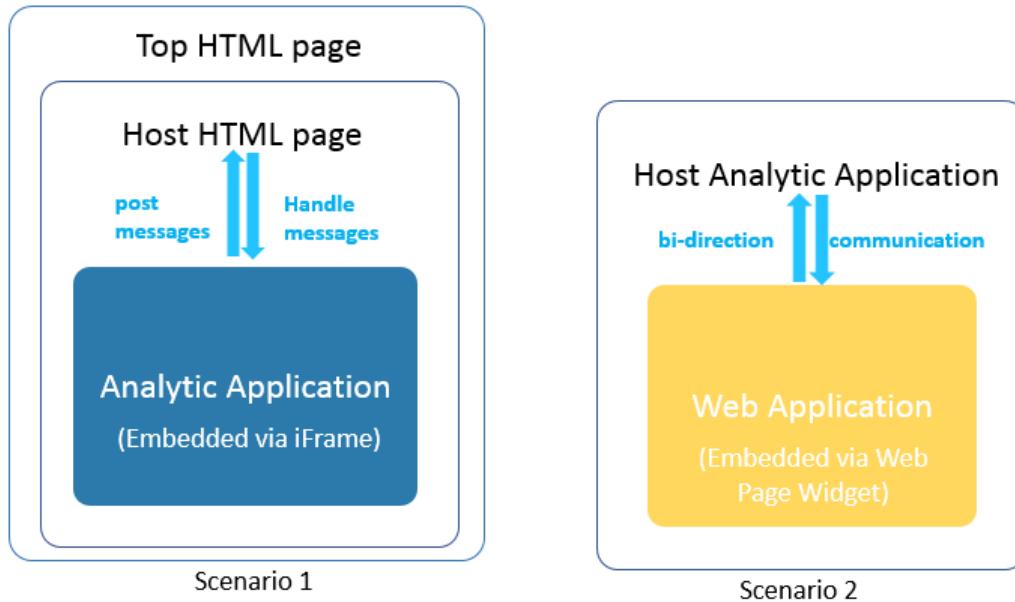


Figure 109: Post Message Scenarios

10.1 Scenario 1: How You Can Embed an Analytic Application in a Host HTML Page via iFrame

Before embedding an analytic application via an iFrame in the host HTML page, you need to first make sure the host HTML page is added as a trust origin in the **System Administration App Integration Trusted Origin**.

Then you can trigger bi-directional communication between the host HTML page and analytic application using the provided functions.

10.1.1 postMessage

This is to post messages from the analytic application to the host HTML page.

When an end user triggers a callback function on the side of the analytic application, the callback function sends out data to notify the parent receiver page which hosts the iFrame, or, when there are multiple levels of web pages embedded in one another, to the top-level HTML page of a specific target origin.

You define whether to send data to a parent or the top HTML page by means of the parameter of the PostMessageReceiver.

The syntax of the postMessage event is:

```
postMessage(receiver: PostMessageReceiver, message: string, targetOrigin: string):  
void
```

10.1.2 onPostMessageReceived

This is to handle messages sent from the host parent or top HTML page in the analytic application. In scenario 2 depicted below, the event can also handle messages sent from an HTML page embedded via the web page widget in an analytic application.

Note: We advise you always to check the origin when receiving an event-triggered message, because a malicious site can change the location of the window and therefore intercept the data you sent using the postMessage event without your knowledge.

In the current scenario, the parent window which hosts the iFrame can post messages to the analytic application's iFrame window of specific target origin. The messages posted are then retrieved by the analytic application and trigger changes accordingly, such as updating some input data.

The syntax of the `onPostMessageReceived` event is:

```
onPostMessageReceived(message: string, origin: string)
```

10.1.3 Example

You can embed an analytic application in a host HTML page. The URL of the host HTML page is <https://localhost:8080>.

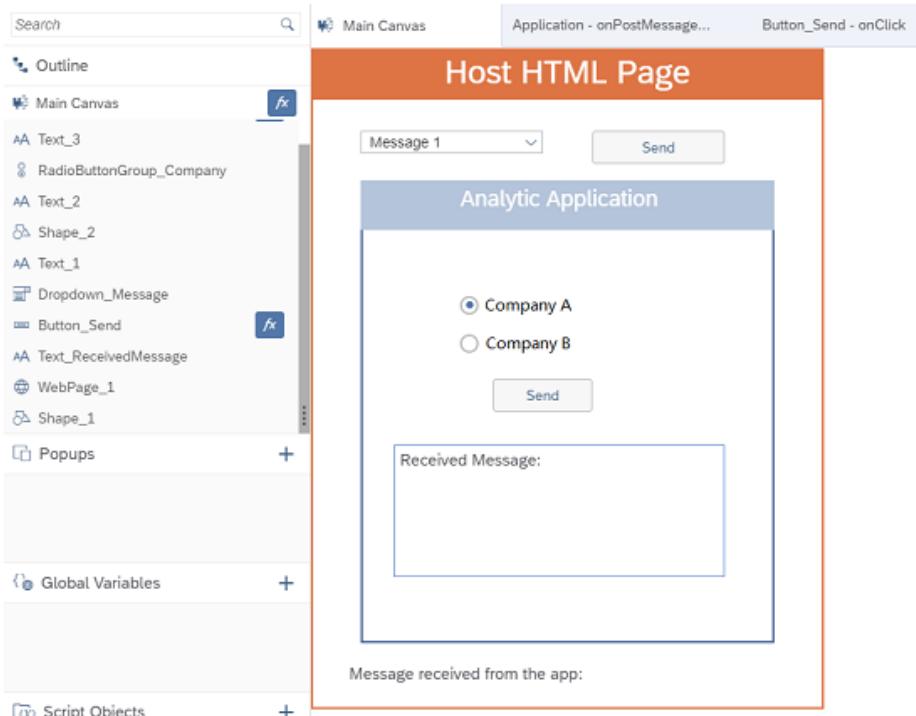


Figure 110: Embed an Analytic Application into a Host Page

First, you want to allow end users to post the company selection in the analytic application to the host HTML page. Write the script below for the sending button:

```
var message = RadioButtonGroup_Company.getSelectedText();
Application.postMessage(PostMessageReceiver.Parent, message,
"https://localhost:8080");
```

Then you want to allow end users to display the message received from the Host HTML page in a text box of the embedded analytic application.

```
if (origin == "https://localhost:8080") {
    Text_ReceivedMessage.applyText(message);
}
```

10.2 Scenario 2: How You Embed a Web Application in an Analytic Application Through the Web Page Widget

You can trigger bi-direction communication between the embedded web application and the host analytic application.

10.2.1 Web Page Widget-Related postMessage and onPostMessageReceived

When the host analytic application's web page widget embeds a web application, you can post messages from the embedded application to the host analytic application or the other way around.

The syntax of the `postMessage` command is:

```
postMessage(message: string, targetOrigin: string): void
```

Note: The target origin is optional. If it's left empty, the URL defined in the web page widget will be taken as the target origin by default.

The syntax of the `onPostMessageReceived` event is:

```
onPostMessageReceived(message: string, origin: string)
```

10.2.2 Case 1 – Posting Messages from the Host Analytic Application to the Embedded Application

The command for posting messages is:

```
postMessage()
```

The event for handling messages sent from the host analytic application depends on the type of the embedded application:

- If the embedded application is an analytic application, once the message is received, the embedded application can use the `onPostMessageReceived()` event to handle the message.
- If the embedded application is another web application, once the message is received, the embedded application can use the event `window.on("message")` to handle the message.

10.2.3 Case 2 – Posting Messages from the Embedded Application to the Host Analytic Application

The event for posting messages depends on the type of the embedded application:

- If the embedded application is an SAP Analytics Cloud application, use the command `Application.postMessage()` to post messages.
- If the embedded application is another web application, use the command `window.parent.postMessage` to post messages.

The event for handling messages sent from embedded application is as follows: Once the messages is received, the host application can use the `onPostMessageReceived()` event to handle the messages.

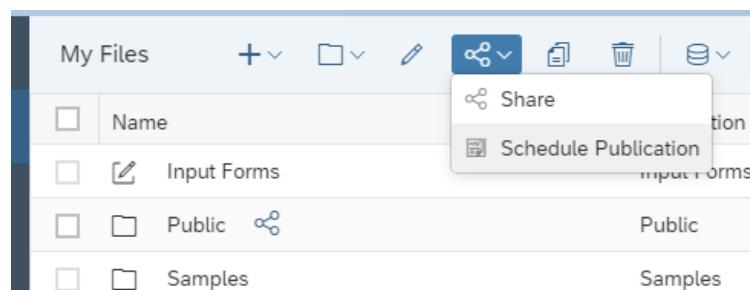
11 Scheduling a Publication

To share an analytic application with other users as well as external users via e-mails at a specific start time and with a specific recurrence, you can schedule an analytic application for publication. Recipients will receive an e-mail with a link to the analytic application (if so configured) and an exported PDF file as an attachment.

11.1 Scheduling a Publication

There are the following places in the user interface to schedule a publication:

- In the *File Repository*



- During the runtime of the application



The *Schedule Publication* dialog is be able to configure the following:

- Publication task name
- Publication start time and recurrence
- E-mail subject and content
- Whether to include the analytic application link in the e-mail or not
- Values of script variables, which will be applied to both the analytic application link included in the e-mail and to the exported PDF file.

Schedule Publication

*Name
New Analytic Application(2)

Start
Mar 24, 2020 - 02:33 PM

+ Add Recurrence

*Subject
Enter e-mail subject

Message

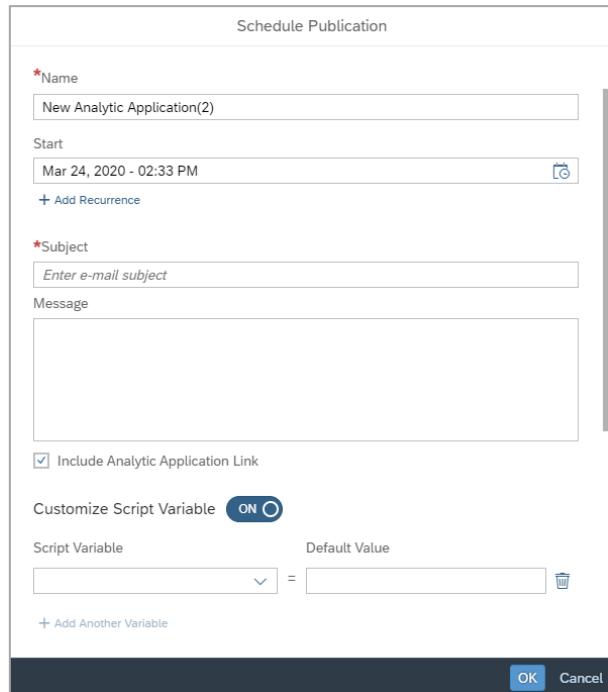
Include Analytic Application Link

Customize Script Variable ON

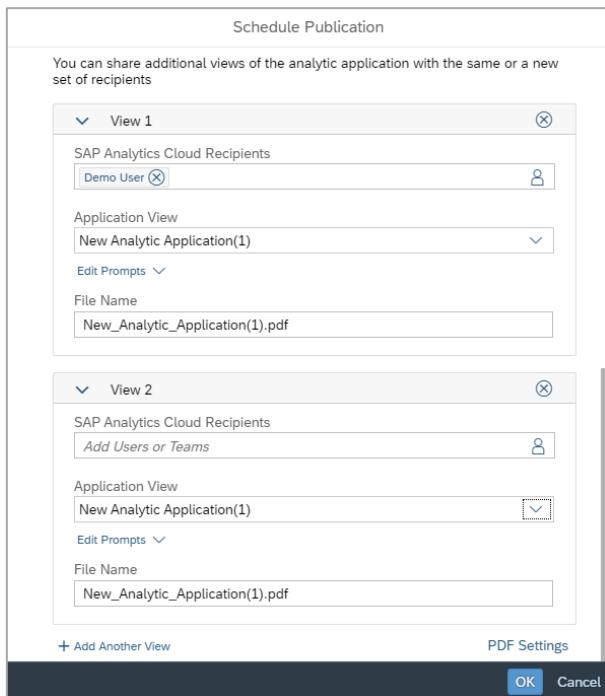
Script Variable	Default Value
<input type="text"/>	<input type="text"/> = <input type="button" value="Delete"/>

+ Add Another Variable

OK Cancel



- Distribution view (at least one view should be configured)
 - Recipients (both SAC users and external users)
 - The application view that's to be exported to a PDF file. This could be the original application document or a bookmark of this document.
 - Edit prompts if necessary
 - File name of exported PDF file
 - PDF settings, such as the following:
 - Paper size
 - Whether to include appendix or not
 - Whether to include comments or not



Once a publication is scheduled, the related task will be available in *Calendar*. The *Design* panel of this task will have the same configurations as those in *Schedule Publication* dialog. The configurations are editable if the task isn't complete, and read-only if it's done.

11.2 Working with System Configurations

There are some configurations under *System Configuration* related to scheduling a publication:

- If *Allow Schedule Publication* is off, then the menu entries in *File Repository* and in the analytic application at runtime will be hidden.
- If *Allow Schedule Publication for non-SAC users* is off, then there will be no user interface to set *Non-SAP Analytics Cloud Recipients* in the *Schedule Publication* dialog.

System Configuration		Datasource Configuration	Security	R Configuration
Name	Value			
Allow Schedule Publication for non-SAC users	<input checked="" type="radio"/> ON <input type="radio"/>			
Allow Schedule Publication	<input checked="" type="radio"/> ON <input type="radio"/>			
Trace Level	Error			
Default Application	SAP Analytics Cloud			

11.3 Script API

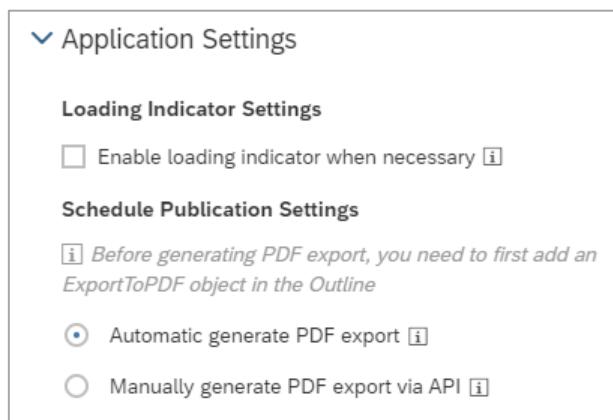
Besides scheduling a publication via the user interface, you can accomplish the same functionality with the following script API:

```
// Integrate with scheduling service and publish pdf  
Scheduling.publish();  
  
// Return whether the application is opened by schedule publication or not  
Scheduling.isRunBySchedulePublication();
```

11.4 Scheduling Publication Settings

There are two types of PDF file generation:

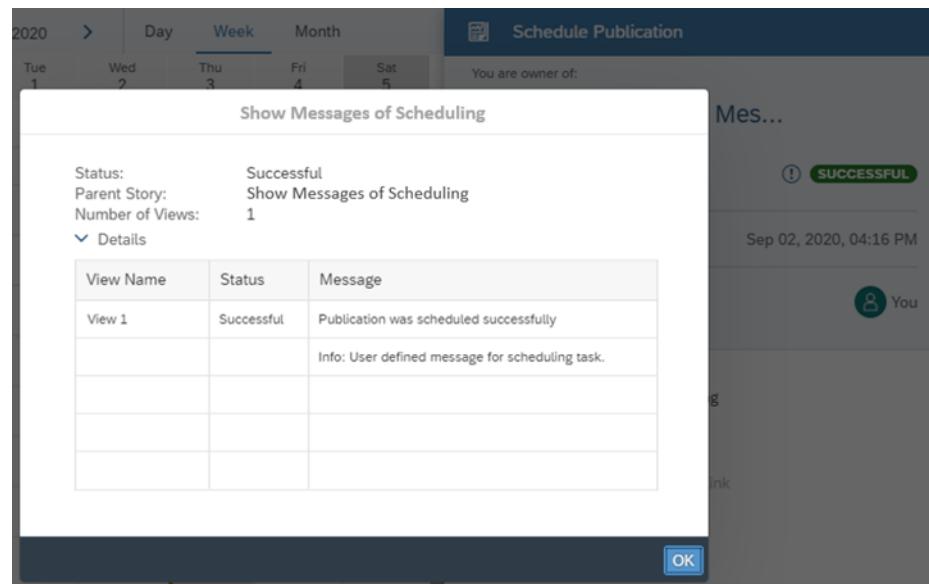
- *Automatically generate PDF export* by a scheduled time when scheduling a publication. The scheduling will be automatically triggered after the `Application.onInitialization` event.
- *Manually generate PDF export via API* using, for example, `Scheduling.publish()`.



11.5 Showing Messages for Scheduling Task

Besides the general status of a scheduling task in the *Schedule Publication* panel in the *Calendar* view, you can get more system runtime messages (for example, details of a failure or error) and user defined messages when an analytic application is run by scheduling.

In the example screenshot below, after clicking in the *Schedule Publication* panel, a dialog of message details pops up. The first row in the table is the system message generated by this scheduling task. The second row is a user-defined message, which adds additional information related to this scheduling task.



You, as an analytic application developer, can manually define the message using the following script API:

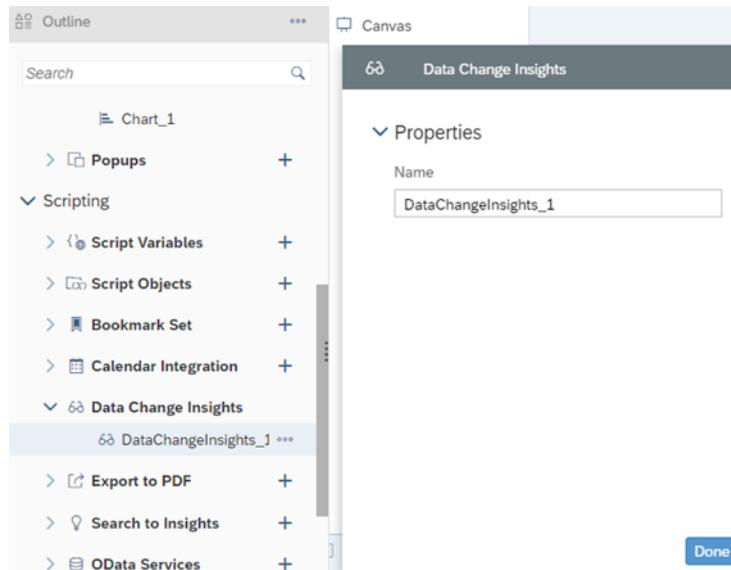
```
Scheduling.logMessage(messageType: SchedulingMessageType, messageText: string)
enum SchedulingMessageType {
    Error,
    Info,
    Warning
}
```

12 Data Change Insights

With Data Change Insights, you, as an analytic application end user, can subscribe to a chart's top N data change insights of an application on a daily, weekly, or monthly basis to intelligently auto-discover significant data changes of your dashboard application within a particular period. You receive the data change insights as a notification via any combination of the following channels: SAC notification, email, or mobile app notification.

In addition, you, as an analytic application developer, can use the Data Change Insights script API to build a Data Change Insights-related application according to your needs.

To start using the Data Change Insights script API, add a *Data Change Insights* component by navigating to *Outline > Scripting > Data Change Insights*, then clicking the + (plus sign).



12.1 Saving a Snapshot

The following Data Change Insights script API saves a Data Change Insights snapshot:

```
DataChangeInsights_1.saveSnapshot();
```

Only one snapshot is kept per analytic application per day. When you create more snapshots within that time frame, then the later snapshot overwrites the earlier one.

The API of manual saving snapshot works only when run by backend scheduling, which is specifically used for some onInitialization scripting scenarios.

12.2 Listing Snapshots

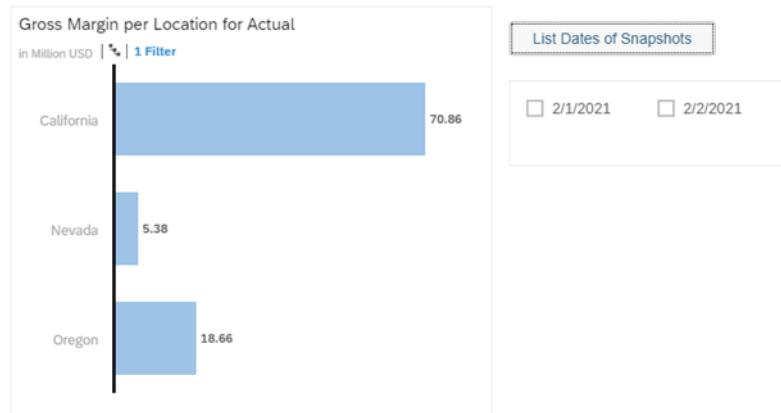
With the Data Change Insights script API, you can retrieve the dates when the most recent Data Change Insights snapshots of the analytic application were saved. By default, up to 10 dates are returned.

Example:

The following sample code retrieves an array of snapshot dates, saves that array in script variable `SV_Snapshots`, then populates a checkbox group with the dates:

```
var SV_Snapshots = DataChangeInsights_1.listRecentSnapshotDates();
for (var i = 0; i < SV_Snapshots.length; i++) {
    CheckboxGroup_1.addItem(i.toString(), SV_Snapshots[i].toLocaleDateString());
}
```

The result below shows that there are two snapshots: one from February 1, 2021 and one from February 2, 2021.



12.3 Comparing Snapshots

With the Data Change Insights script API, you can compare Data Change Insights snapshots in two different ways:

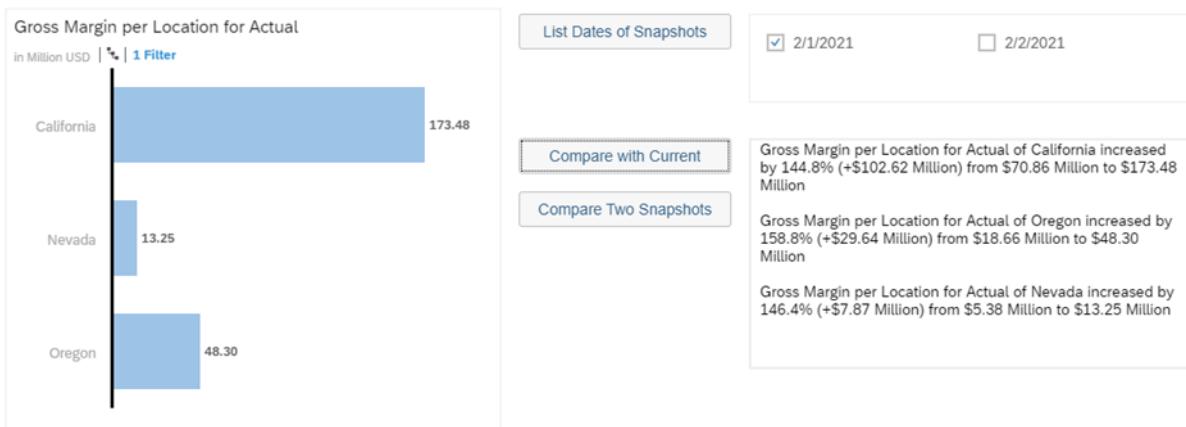
1. Compare a snapshot with the current application state
2. Compare two snapshots (by date)

Example: (Comparing a snapshot with the current application state)

The following sample code compares the current application state with the first selected snapshot of the checkbox group and displays the comparison result in the text area:

```
var selection = CheckboxGroup_1.getSelectedKeys();
if (selection.length > 0) {
    var snapshotIndex = ConvertUtils.stringToInteger(selection[0]);
    var result = DataChangeInsights_1.compareApplicationStateWithSnapshot(
        SV_Snapshots[snapshotIndex]);
    if (result.status === DataChangeInsightsStatus.Ok) {
        var insights = result.insights;
        if (insights.length > 0) {
            var contents = ArrayUtils.create(Type.string);
            for (var i = 0; i < insights.length; i++) {
                var description = insights[i].content;
                description = description.split("<keyword>").join("");
                description = description.split("</keyword>").join("");
                contents.push(description);
            }
            TextArea_1.setValue(contents.join("\n\n"));
        }
    }
}
```

The result is shown below in the text area: The gross margin per location changed between February 1, 2021 and today.



Example: (Comparing two snapshots with each other)

The following sample code compares the first and second selected snapshots of the checkbox group and displays the comparison result in a text area:

```

var selection = CheckboxGroup_1.getSelectedKeys();
if (selection.length > 1) {
    var sourceIndex = ConvertUtils.stringToInteger(selection[0]);
    var targetIndex = ConvertUtils.stringToInteger(selection[1]);
    var result = DataChangeInsights_1.compareSnapshots(SV_Snapshots[sourceIndex],
    SV_Snapshots[targetIndex]);
    if (result.status === DataChangeInsightsStatus.Ok) {
        var insights = result.insights;
        if (insights.length > 0) {
            var contents = ArrayUtils.create(Type.string);
            for (var i = 0; i < insights.length; i++) {
                var description = insights[i].content;
                description = description.split("<keyword>").join("");
                description = description.split("</keyword>").join("");
                contents.push(description);
            }
            TextArea_1.setValue(contents.join("\n\n"));
        }
    }
}

```

The result is shown below in the text area: The gross margin per location changed between February 1, 2021 and February 2, 2021.

The screenshot shows a user interface for comparing snapshots. At the top left is a button labeled "List Dates of Snapshots". To its right are two date inputs: "2/1/2021" and "2/2/2021", each preceded by a checked checkbox. Below these are two buttons: "Compare with Current" and "Compare Two Snapshots". To the right of these buttons is a large text area containing three lines of comparison results:

- Gross Margin per Location for Actual of California increased by 76.1% (+\$53.95 Million) from \$70.86 Million to \$124.81 Million
- Gross Margin per Location for Actual of Oregon increased by 83.9% (+\$15.65 Million) from \$18.66 Million to \$34.31 Million
- Gross Margin per Location for Actual of Nevada increased by 68.7% (+\$3.69 Million) from \$5.38 Million to \$9.07 Million

12.4 Configuring Snapshot Storage

By default, Data Change Insights snapshots are saved in the local tenant storage. You can configure the data repository where snapshots are stored by selecting navigating to *System > Administration*, selecting the *Datasource Configuration* tab, then moving to the *Data Change Insights Snapshots* section.

The screenshot shows the "Datasource Configuration" tab selected in the navigation bar. Below the navigation bar is a toolbar with icons for edit, refresh, and save. A button labeled "+ Add a Data Repository" is visible. The main content area is titled "Data Change Insights Snapshots" and contains the following text:

Configure where to store your data change insights snapshots. The storage location is the local tenant by default. If you are a live data connection user, and want to save data changes to your own HANA system, you can also choose any other data repository from the list, which is configured in Data Repositories.

Storage to Save Snapshots:

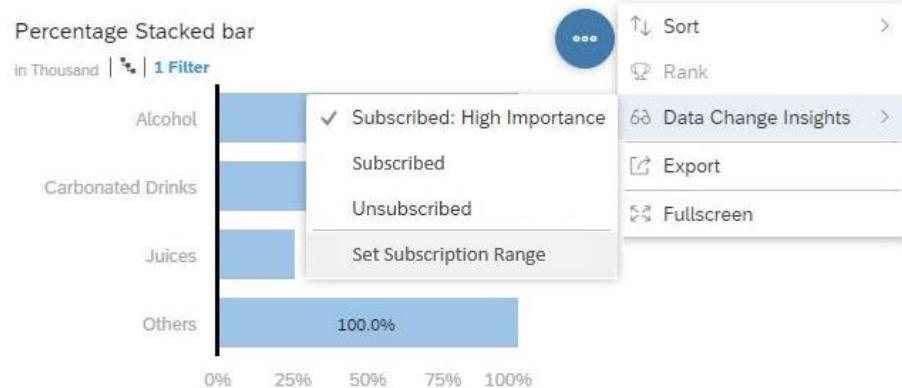
DataChangeInsightsRemoteSnapshot

To configure the data repository where snapshots are stored, you need to have the *Remote Repository Snapshot privilege create*.

<input type="checkbox"/>	Name  	<input type="checkbox"/> Create	<input checked="" type="checkbox"/> Read	<input type="checkbox"/> Update	<input type="checkbox"/> Delete	<input type="checkbox"/> Execute	<input type="checkbox"/> Maintain	<input type="checkbox"/> Share	<input type="checkbox"/> Manage
<input type="checkbox"/>	> Custom Widget	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
<input type="checkbox"/>	Private Application Bookm...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	
<input type="checkbox"/>	Validation Rule	<input type="checkbox"/> Plannin...	<input type="checkbox"/>	<input type="checkbox"/> Plannin...	<input type="checkbox"/> Plannin...				
<input type="checkbox"/>	Publish Content					<input type="checkbox"/>			
<input type="checkbox"/>	Catalog Administration							<input type="checkbox"/>	
<input type="checkbox"/>	Content Link	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
<input type="checkbox"/>	Synonym Dictionary	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
<input checked="" type="checkbox"/>	Remote Repository Snapshot	<input checked="" type="checkbox"/>							
<input type="checkbox"/>	Runtime Notification	<input checked="" type="checkbox"/>							

12.5 Setting Subscription Levels and Ranges

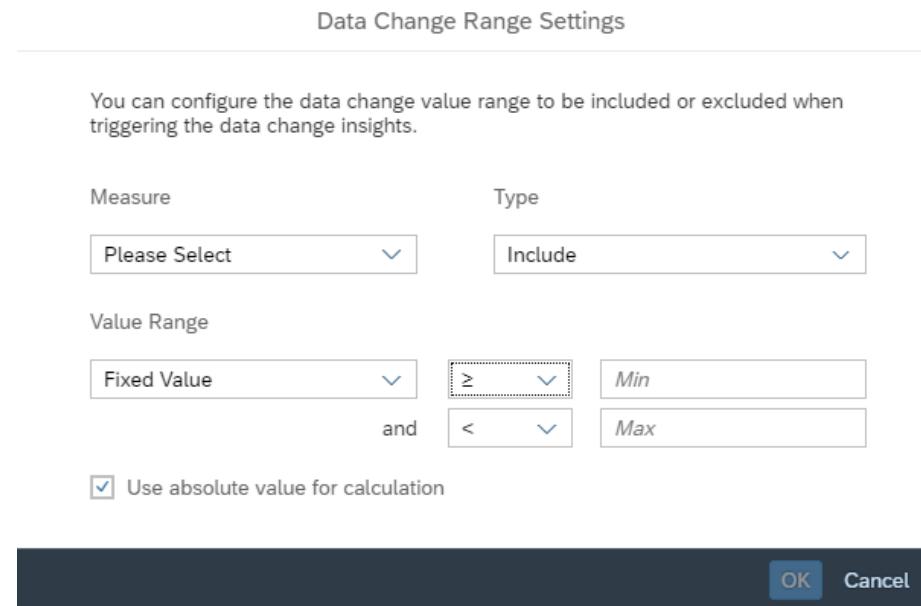
At runtime, you can set the chart level change range with the context menu item *Data Change Insights*. This lets you control whether a data point value is to be included during the Data Change Insights calculation or not.



In the submenu you can select a subscription level from the following values:

- *Subscribed: High Importance*
- *Subscribed*
- *Unsubscribed*

You can define a subscription range with the *Data Change Range Settings* dialog:



It lets you configure the following properties:

- *Measure*
You can specify the measure or account.
- *Type*
You can include or exclude the range during the calculation.
- *Value Range*
You can specify the value range to be either a fixed value or a delta value as well as a minimum and maximum value, including operators like <, <=, >, and >=.
- *Use absolute value for calculation*
You can specify whether to use an absolute value for the calculation or not.

12.5.1.1 Script API

You can get the Data Change Insights object with the following script API:

```
Chart.getDataChangeInsights(): DataChangeInsights
```

The following script API opens the subscription dialog:

```
DataChangeInsights.openSubscriptionDialog(): void
```

The following script API lets you configure the subscription level:

```
DataChange .setSubscriptionLevel(level: DataChangeInsightsSubscriptionLevel)
DataChangeInsights.getSubscriptionLevel(): DataChangeInsightsSubscriptionLevel
```

where the subscription level is one of the following values:

```
DataChangeInsightsSubscriptionLevel.SUBSCRIBE
DataChangeInsightsSubscriptionLevel.SUBSCRIBE_HIGH_IMPORTANCE
DataChangeInsightsSubscriptionLevel.UNSUBSCRIBE
```

The following script API lets you configure subscription ranges:

```
DataChangeInsights.setSubscriptionRange(range: DataChangeInsightsSubscriptionRange)
DataChangeInsights.getSubscriptionRange(): DataChangeInsightsSubscriptionRange
```

where a subscription range has the following properties:

```
class DataChangeInsightsSubscriptionRange {
```

```

        structure: string,
        structureMember: string,
        min: number,
        max: number,
        isInclude: boolean, // optional (default: false)
        isDeltaValue: boolean, // optional (default: false)
        isMinOrEqual: boolean, // optional (default: false)
        isMaxOrEqual: boolean, // optional (default: false)
        isAbsoluteValue: boolean // optional (default: false)
    }
}

```

Example:

In the following examples, several subscription range settings are shown for different data models:

Model with Accounts

```

var DataChangeInsightsSubscriptionRange_1 = {
    structure: "Account_BestRunJ_sold",
    structureMember: "[Account_BestRunJ_sold].[parentId].&[Gross_Margin]"
    min: 0,
    max: 10
}

```

Model with Measures

```

var DataChangeInsightsSubscriptionRange_2 = {
    structure: Alias.MeasureDimension,
    structureMember: "LOTAMOUNT_Q42021"
    min: 0,
    max: 10
}

```

Model with Accounts and Measures

```

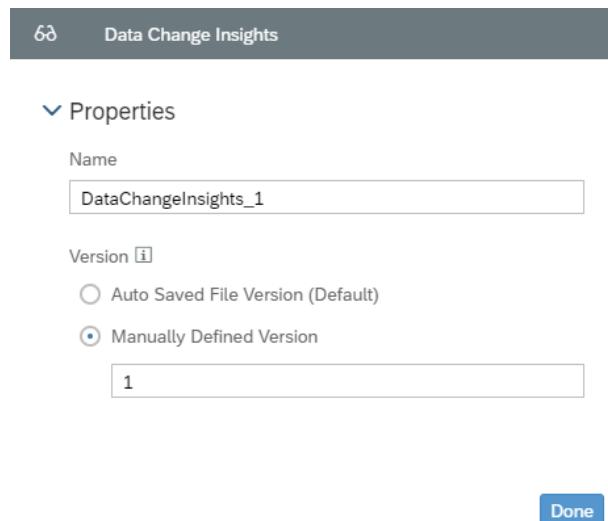
var DataChangeInsightsSubscriptionRange_3 = {
    structure: "TST_ACCOUNT",
    structureMember: "[TST_ACCOUNT].[parentId].&[COGS]"
    min: 0,
    max: 10
}

```

12.6 Setting and Getting the Version

A Data Change Insights component can specify a version for snapshot validation. A snapshot is valid if the snapshot's version is equal or greater than the version of the Data Change Insights component.

When you add a Data Change Insights component, a *Data Change Insights* panel opens:



In this panel you can set the component's version in the following ways:

- *Auto Saved File Version (Default)* – The version is specified by the timestamp taken when the analytic application is saved.
- *Manually Defined Version* – You can specify the version by manually entering a version number.

12.6.1.1 Script API

The following script API lets you retrieve the version of the Data Change Insights component:

```
DataChangeInsights.getVersion(): integer
```

13 Blend Data in Analytic Applications

First of all, blending is available in both analytic applications and stories. This feature is same in analytic applications and stories, except for specific widgets and script APIs in analytic applications. If you would like to learn more background information, prerequisites and steps about blending, refer to [Blending Data](#).

In a nutshell, using the data blending, you can join a primary model with secondary ones that contain common dimensions in your analytic applications. It is very useful to create visualizations with blended data source, as well as filters that can simultaneously update all the charts and tables with linked dimension from different data sources, without writing any script.

It worth checking the restrictions for advanced usage of data blending.

Restrictions

All the restrictions in stories are applicable to analytic applications as well. Also refer to Blending Data.

In addition, the following features in analytic applications are not yet supported for blending:

- universal model
- filter line
- data point comment
- navigation panel

Notes about Scripting

- Currently, the APIs and value help only work on primary model of blended chart or table, while secondary models aren't supported. For example, you can't use setDimensionFilter to filter dimension members in a secondary model.
- For blended charts or tables, the IDs of dimensions and measure members contain modelId. Therefore, when you need to use the output from one API as the input parameter for the consequent one, remember to check whether the dimension IDs in the two APIs match.
- Calculated measures don't belong to either primary or secondary models, so they don't have modelId but UUID.
- It is a limitation that the value help for blended charts or tables doesn't return modelID. We suggest printing the chart or table selection to the browser console to get the full ID.

Example:

You'd like to use the product selections in Table_3, which is a blended table, to filter data in Table_1, which uses a single model. The value help for the product dimension doesn't include the modelID, so you need to use console.log to get the right ID.

```
var sel = Table_3.getSelections();
console.log(sel);
var products = ArrayUtils.create(Type.string);
for (var i = 0; i < sel.length; i++){
    // use keyboard shortcut: CTRL+Space will bring up the value help for Product
    // dimension, but it doesn't provide the complete id which should include the modelId.
```

```
// products.push(sel[i]["0D_FC_PH2"]);
// the correct one
products.push(sel[i]['@{["t.H:C3X4VAAY0IWGHRZLRCHLA67DVK","0D_FC_PH2"]}]']);
}
Table_1.getDataSource().setDimensionFilter("0D_FC_PH2", products);
```

Unsupported APIs

The following APIs are not yet supported for blending:

- All APIs related to planning
- All APIs related to navigation panel
- All APIs related to comment
- All APIs related to data explorer
- Chart.getDataSource().SetDimensionFilter(), Chart.getDataSource().RemoveDimensionFilter()
- DataSource.copyDimensionFilterFrom(), DataSource.copyVariableValueFrom()
- Table.sortByMember(), Table.sortByValue(), Table.removeSorting()
- Table.rankBy(), Table.removeRanking()
- Table.setBreakGroupingEnabled()

14 Performance Best Practices

This section provides best practices for improving the performance of your analytic application. Some best practices are universally applicable. Some apply only to specific situations. Some need to be tried out with your actual analytic application at hand to determine the degree of performance improvement.

14.1 Top Picks

The following best practices can improve the performance of analytic applications significantly. They're regularly recommended to analytic application developers by SAP Development Support.

- [Loading Invisible Widgets in Background](#)

Improve the perceived startup time of analytic applications with several pages (or tabs) but few widgets visible on the first page (or tab) by having the invisible widgets loaded in the background.

- [Use the Pause Refresh API](#)

Control when to refresh the data source of your chart or table – also in planning scenarios – thus reducing the number of roundtrips to fetch data from the backend.

- [Enable Planning on Tables Only When Planning is Used](#)

Decide whether and when to enable planning on tables.

- [Use MemberInfo Object with setDimensionFilter\(\)](#)

Supply a description when setting a filter to avoid roundtrips to fetch the description from the backend.

14.2 Analytic Application Script Performance Popup

You, as an analytic application developer, can analyze the performance of your analytic application, in particular its script performance, by using the Analytic Application Script Performance Popup. It visualizes the duration of your script execution and operations which influence script execution performance in a negative way.

14.2.1 Basic Steps

14.2.1.1 Opening the Popup

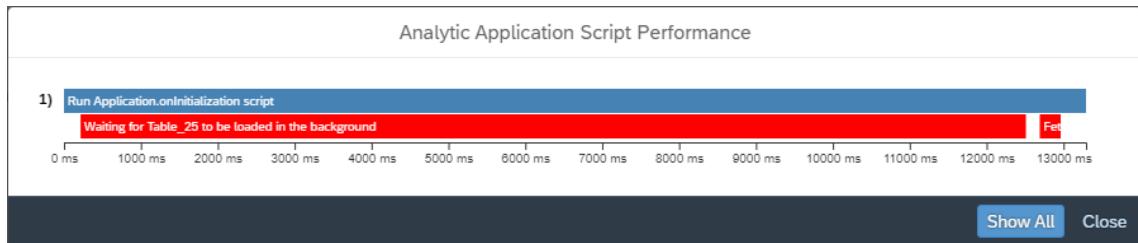
To enable the Analytic Application Script Performance Popup, do either of the following:

- At runtime, enter the parameter `APP_PERFORMANCE_LOGGING=true` in the application URL. Enter the parameter in front of the `#` character in the URL and precede the parameter with `?`.
- From the design time toolbar, select *Performance Optimization->Analyze Script Performance in Runtime*.

The application will be run in a new tab, with the parameter added to the URL.

To open the Analytic Application Script Performance Popup, when the application is fully loaded, press `Ctrl+Shift+A` or `Ctrl+Shift+Z`.

In the following screenshot you see an example of the Analytic Application Script Performance Popup:



A blue bar visualizes the script execution time of the `onInitialization` event script (ca. 13,000 ms).

Two red bars visualize operations which influence script execution performance in a negative way – and which can be avoided by changes to the script. In this example they represent (1) waiting for a specific table to be loaded in the background and (2) fetching the description of a dimension member.

14.2.1.2 Using the Popup

The Analytic Application Script Performance Popup keeps a record of the script executions during the entire session of the analytic application. For example, when you modify the page in analytics designer and view the page again, another blue bar is added below the previous one such that you can compare your efforts to optimize the script execution time.

To get a tooltip displaying the execution time, hover over a blue or red bar.

To hide a bar, click on it. The remaining bars are rescaled to fill the width of the Analytic Application Script Performance Popup.

To restore all bars, click the `Show All` button.

14.2.2 Sample Optimization Walkthrough

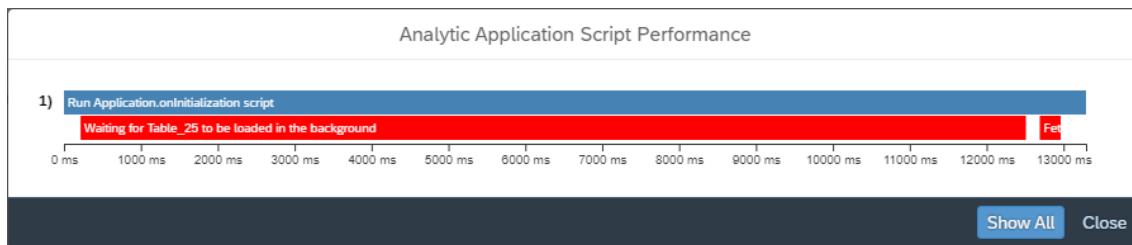
Let's assume, you've an analytic application with 25 tables. They're all planning-enabled and their visibility is set to false. The analytic application has already activated loading invisible widgets in the background (see [Loading Invisible Widgets in Background](#)).

The purpose of the analytic application is to set a filter, then to retrieve the description of a member of Table 25's data source and to assign this description to a text field.

This operation is performed with the following `onInitialization` event script:

```
var ds = Table_25.getDataSource();
ds.setDimensionFilter("ResponsibilityCenter",
{ id: "[ResponsibilityCenter].[parentId].&[HREG0001]" });
Text_1.applyText(ds.getMeasures()[0].description);
```

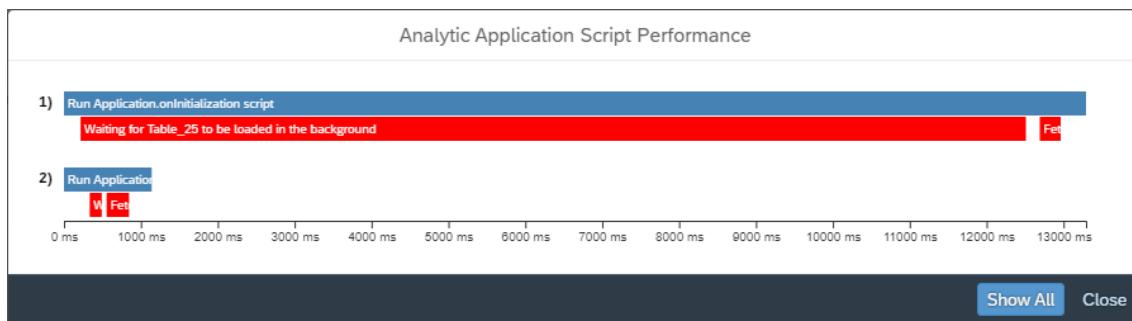
When you run the analytic application and then open the Analytic Application Script Performance Popup, the following picture is shown:



Its red bars reveal that the `onInitialization` event script is mostly waiting for loading Table 25 in the background (large red bar) and for fetching a description (small red bar).

14.2.2.1 Optimization 1 – Using *Always initialize on startup* When Accessing an Invisible Widget in `onInitialization`

In analytics designer, for invisible Table 25, navigate in the *Styling* panel to *Actions* and enable *Always Initialize on startup* (see [Loading Widgets in Background – Background Knowledge and Tips](#)). Run the analytic application again, then open the Analytic Application Script Performance Popup:



The Analytic Application Script Performance Popup shows that the script runs much faster than before. But still, the analytic application spends some time waiting for Table 25 and for fetching the description.

14.2.2.2 Optimization 2 – Using Pause Refresh

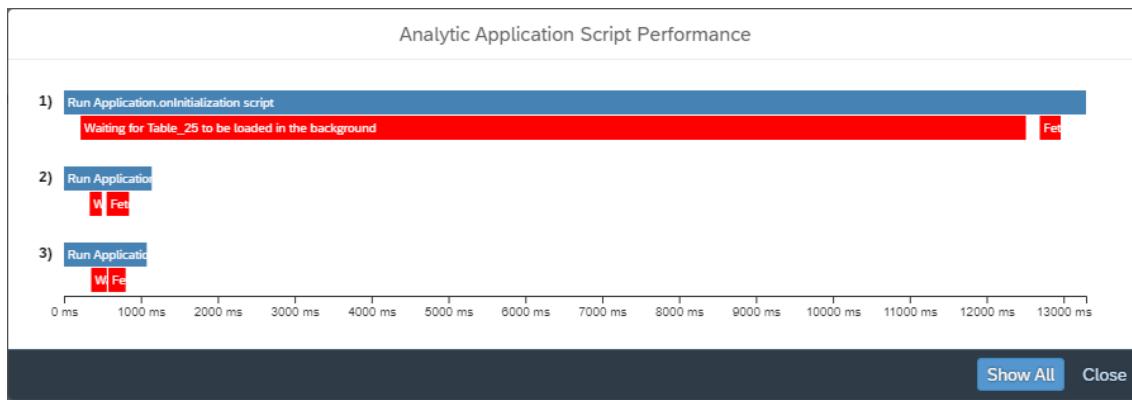
Table 25 is now initialized at startup, but then it's modified again by setting a filter in the `onInitialization` event script, which makes it necessary to fetch the result set yet again. Let's use Pause Refresh (see [Use the Pause Refresh API](#)) to prevent fetching the result set two times.

In analytics designer, for Table 25, in the *Builder* panel, navigate to *Properties* and enable *Pause Data Refresh*. Note: This automatically disables *Planning Enabled*.

Then, adjust the `onInitialization` event script as follows:

```
var ds = Table_25.getDataSource();
ds.setDimensionFilter("ResponsibilityCenter",
  { id: "[ResponsibilityCenter].[parentId].[HREG0001]" });
ds.setRefreshPaused(false);
// Next line only necessary if planning is used directly after analytic application
// startup
Table_25.getPlanning().setEnabled(true);
Text_1.applyText(ds.getMeasures()[0].description);
```

Run the analytic application again, then open the Analytic Application Script Performance Popup:



In the Popup above there is still a first red bar representing a few milliseconds of waiting time. In other runs of the analytic application, this bar may disappear altogether. It's present here because the execution of the `onInitialization` event script starts and the creation of the initially rendered widgets hasn't fully completed.

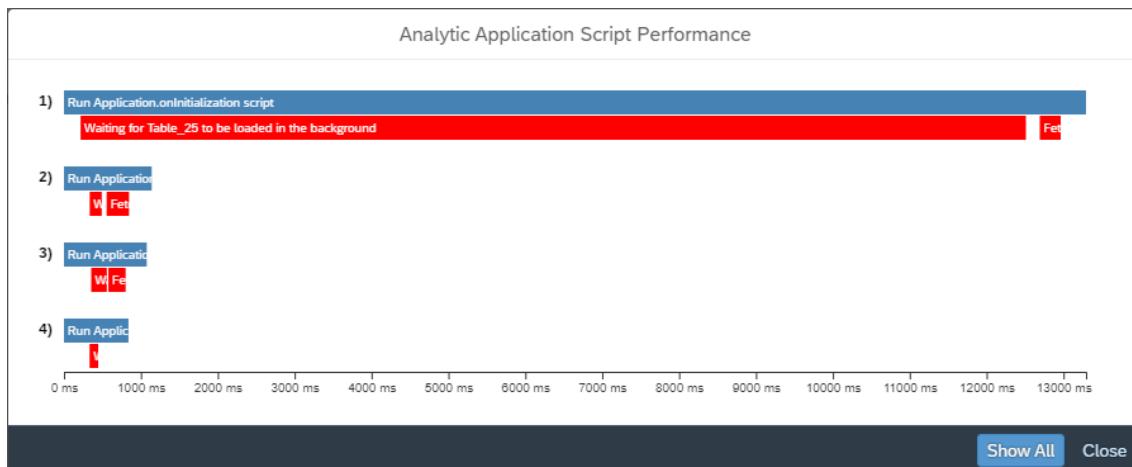
Now let's tackle the second red bar, representing the remaining waiting time for fetching the description.

14.2.2.3 Optimization 3 – Passing a Description to setDimensionFilter()

In analytics designer, adjust the `onInitialization` event script once more by passing a dummy description (see [Use MemberInfo Object with setDimension](#)):

```
var ds = Table_25.getDataSource();
ds.setDimensionFilter("ResponsibilityCenter",
{ id: "[ResponsibilityCenter].[parentId].&[HREG0001]",
  description: "mydescription" });
ds.setRefreshPaused(false);
// Next line only necessary if planning is used directly after analytic application
// startup
Table_25.getPlanning().setEnabled(true);
Text_1.applyText(ds.getMeasures()[0].description);
```

Run the analytic application again, then open the Analytic Application Script Performance Popup:



The Analytic Application Script Performance Popup shows that the second red bar, representing the waiting time for fetching the description, is gone.

14.2.2.4 Summary

By analyzing the performance of the analytic application with the Analytic Application Script Performance Popup, the execution time of the `onInitialization` event script has been significantly reduced.

14.3 Analytic Applications

The following best practices apply to analytic applications in general.

14.3.1 Stories and Analytic Applications

- Apply best practices documented for stories to analytic applications, as analytic applications are based on stories. This applies also to widgets shared between stories and analytic applications, for example, tables, charts, and geo maps.

14.3.2 Browser

- Use a Google Chrome (or a Chromium-based) browser.
- Take advantage of improved performance with the browser's caching of analytic applications. This is particularly important for apps with multiple charts or models. The cache is valid as long as there are no structural changes made in the analytic application. Note that this performance improvement is only available for Chrome users in a Chrome's non-incognito mode.

14.3.3 Application Design

- Avoid building 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 script 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've 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()` script API method to display key figure values. This improves performance by avoiding multiple requests to the backend.

14.3.4 Mobile Devices

- Reduce the number of elements in your analytic application. This reduces the size and thus the memory consumption of your app on the mobile device. Your mobile device should have 2 GB memory minimum, 3 GB is recommended.

14.3.5 Application Startup 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.

14.4 Data Sources and Widgets

The following best practices apply to data sources and widgets of analytic applications.

14.4.1 Data Sources

- Consider using unbooked data over booked data. This may reduce server processing time.

Note: This sounds counter-intuitive as it violates the best practice of reducing the amount of transferred data as much as possible. However, in some situations, the reduction of server processing time may be larger than the transfer time of an expanded amount of data.

14.4.2 Charts

- Reduce the number of charts. This reduces the amount of transferred data.
- Reduce the number of visible data points.

14.4.3 Tables

- Reduce the number of tables. This reduces the amount of transferred data.
- Limit tables to 500 rows and 60 columns.

14.4.4 Images

- Consider limiting the size of images to less than 1 MB.
- Prefer image file formats in the order: SVG, PNG, JPG.

14.4.5 GeoMaps

- When using a *Bubble Layer*, turn on *Location Clustering*. Enter 1000 for *Maximum Display Points*. Both reduces the amount of transferred data.
- When working with thousands of locations, consider using the *Choropleth Layer*.

14.4.6 Avoid Duplicating Widgets Unnecessarily

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

- Bad pattern (due to restrictions 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()`.

14.4.7 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 script API to move this widget to the currently visible container, for example, a tab of a tab strip or a panel.

14.4.8 Loading Invisible Widgets in Background

This provides a faster perceived startup time of analytic applications that contain (many) invisible widgets.

Introduction

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.

Now, you, as an analytic application developer, can change this default behavior by activating loading invisible widgets in the background.

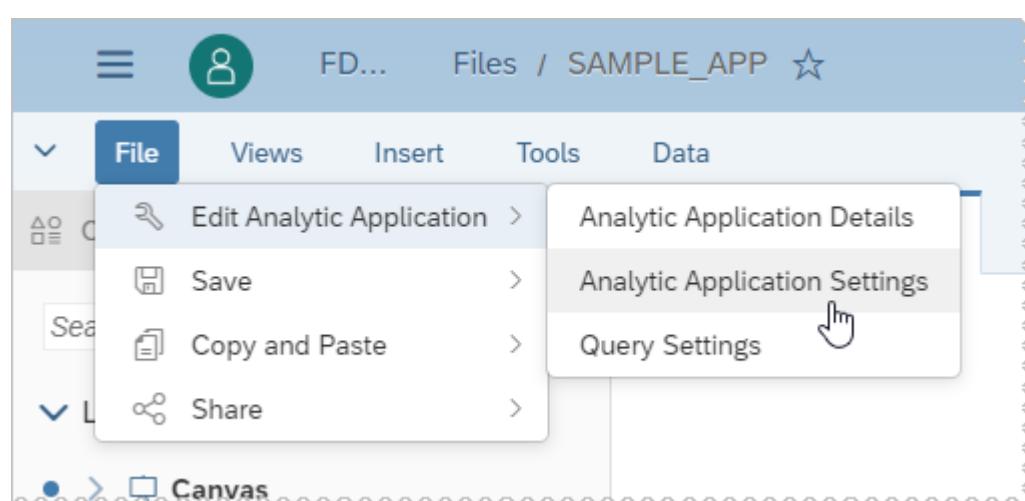
When activated, all widgets that are initially visible are initialized and displayed to the end user. After that, the invisible widgets are initialized in the background to be available when the end user changes their visibility.

Invisible widgets aren't only widgets with the *Builder* panel 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.

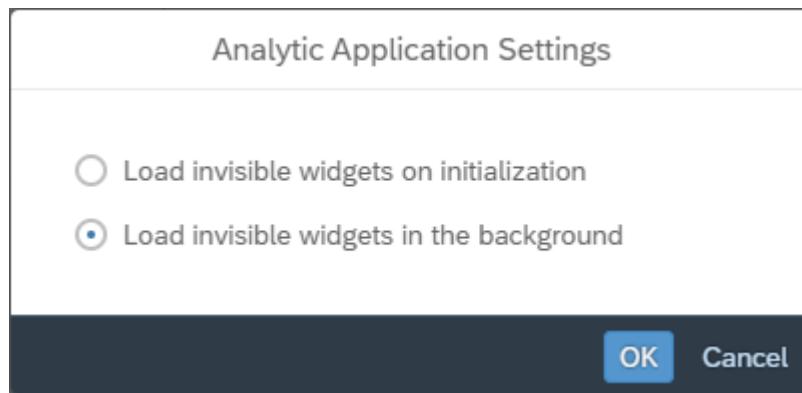
This new loading behavior increases the perceived startup performance of the analytic application because the startup screen appears faster.

Getting Started

You can change to the new behavior 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 analytic application developer, want to decide, which mode is working better with your analytic application or if you, as an end user, aren't satisfied with the choice of the analytic application developer.

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 the visible widgets are displayed to the end user:

```
loadInvisibleWidgets=inBackground
```

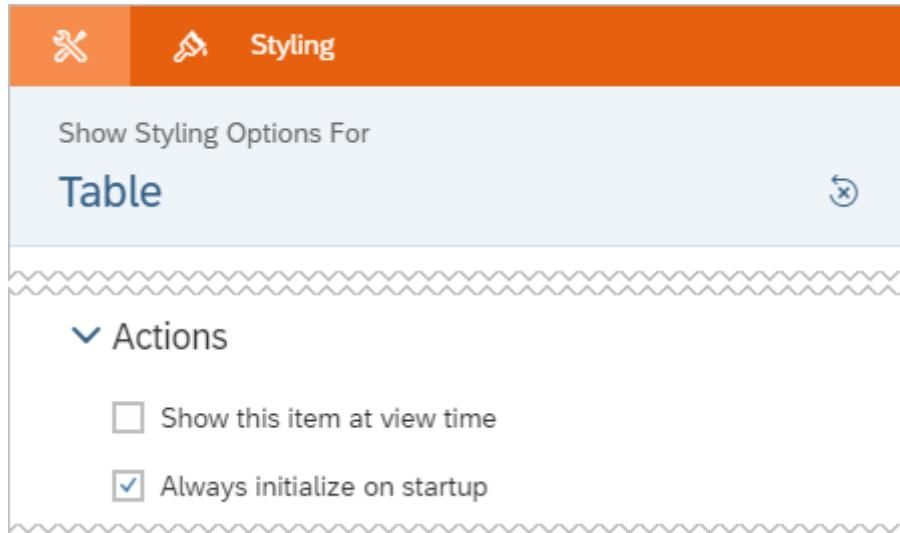
14.4.8.1 Background Knowledge and Tips

If the mode *Load invisible widgets in background* is used, it's possible that a script tries to access a widget which doesn't exist at this point in time. This applies especially to script code in 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 widgets that are initially invisible. Especially avoid using the `setVariableValue()` method if not needed.
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 (including 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's 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's initialized as if it was visible. In addition, its visible child widgets are also initialized.



14.4.8.2 Tip: Consider Device Viewport When Using Loading Widgets in Background

Background loading loads all invisible widgets in the background. Widgets that aren't in the viewport of the device, in other words, widgets for which you need to scroll or swipe to make them appear on the device, are also considered invisible.

When you create an analytic application for a laptop device, then the application remembers the position of the widgets. If you open this application on a mobile device (with a smaller viewport than the laptop), then widgets that are outside of the mobile device's viewport are considered invisible. They are thus separately loaded in the background and may appear after the application has been already displayed initially.

This happens with analytic applications with widgets that are docked to the right or to the bottom of the application (widgets with a *Size and Position* value of *Left (X)* or *Top (Y)* of *auto*).

To avoid this, save your analytic application with the smallest device viewport that is used to display the application. You can select the device viewport with the *Device:* dropdown at the bottom border of the Canvas.

14.5 Scripting

The following best practices apply to scripting in analytic applications.

14.5.1 Group Several `setVariableValue()` Calls with 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 API 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.

14.5.2 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.

14.5.3 Use `getResultSet()` Instead of `getMembers()`

The `getResultSet()` method of a data source can use the result set that's already available in the widget and doesn't need a roundtrip to the backend system. The `getMembers()` method of a data source always leads to an extra roundtrip to the backend system.

If it's sufficient for your scenario you should consider using method `getResultSet()`.

If you need to use method `getMembers()`, then make sure to use the available `options` parameter to limit the list of returned members.

14.5.4 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.

14.5.5 Avoid 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 API methods during the `onInitialization` event, as 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 really need to modify data sources of widgets during the `onInitialization` event, for example, by setting different filters or variable values, consider using the Pause Refresh script API. For more information, see [Use the Pause Refresh API](#).

14.5.6 Initialize Variables Via URL Parameters

You can set (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 information, see the help page of *Variable Parameters*:

<https://help.sap.com/viewer/a4406994704e4af5a8559a640b496468/release/en-US/3fa25603111049fba11eb898ccfabbb3.html>

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>?v01  
Model=view:[_SYS_BIC][t.TEST][remotejuice]&v01Par=Manager&v01Val=["SM1", "SM2"]
```

14.5.7 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.length; i++) {  
    ds.setDimensionFilter(dimensions[i], value);  
}
```

14.5.8 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()`.

14.5.9 Enable Planning on Tables Only When Planning Is Used

You can enable or disable planning for a table at design time in the *Builder* panel of the table in the section *Properties* > *Planning* with the checkbox *Planning Enabled*. This option is only available with models that support planning.

If you add a table that's based on a planning model, then planning is enabled by default.

Disable planning if the table isn't used for planning at all.

Disable planning if the table isn't visible at startup, then apply method `getPlanning().setEnabled()` on the table to enable or disable planning at runtime, depending on whether you need planning or not.

Example:

In the following example, an analytic application has a TabStrip widget with two tabs. The second tab contains a planning-enabled table. Whenever this tab becomes visible, then planning of the table should be enabled. Whenever this tab becomes invisible, then planning of the table should be disabled.

You can accomplish this by the following `onSelect` event script of the TabStrip widget:

```
var selectedTab = TabStrip_1.getSelectedKey();
if (selectedTab === "Tab_2") {
    Table_1.getPlanning().setEnabled(true);
} else {
    Table_1.getPlanning().setEnabled(false);
}
```

Note: When you use planning in combination with Pause Refresh, then Pause Refresh is only available if planning is disabled. For more information, see [Use the Pause Refresh API](#).

14.5.10 Prefer Local Variables over Global Variables

If you want to use a variable only in one script, use a local variable.

14.5.11 Prefer Function Arguments over Global Variables

If you want to use a variable in a script function, pass it as an argument instead of using a global variable.

14.5.12 Use MemberInfo Object with `setDimensionFilter()`

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

```
var memberId = Dropdown_1.getSelectedKey();
Table_1.getDataSource().setDimensionFilter("sap.epm:Department", memberId);
// Roundtrip performed to fetch member description
```

When you pass a `MemberInfo` object (it contains a description) instead of a member ID string, then no roundtrip to the backend is performed:

```
var memberId = Dropdown_1.getSelectedKey();
var memberDescription = Dropdown_1.getText();
Table_1.getDataSource().setDimensionFilter("sap.epm:Department", {id: memberId, description: memberDescription});
// No roundtrip performed to fetch member description. It's already present.
```

You can use this pattern also with an array of `MemberInfo` objects:

```
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);
// No roundtrips performed to fetch member descriptions. They're already present.
```

Apply this pattern whenever the member description is available to you.

Note: If the filter definition is never visible to the end user of your analytic application, simply use a dummy description.

14.5.13 Use the Pause Refresh API

You can use the Pause Refresh API to improve the performance of your analytical application (for more information about the Pause Refresh API, see [Pause Refresh Script API](#)).

A range of situations is described here in detail.

Pause the initial refresh of charts or tables when modifying them in the `onInitialization` event script

Pause the initial refresh of charts or tables until after you've modified them in the `onInitialization` event script, for example, after applying a filter. This improves the analytic application's startup time.

Example:

In the following example, an analytic application contains a table. In the `onInitialization` event script a filter is applied to the table.

1. Enable Pause Refresh for this table in its *Builder* panel section *Properties* > *Pause Data Refresh*.
2. Add the following code to the `onInitialization` event script:

```
// ...
var ds1 = Table_1.getDataSource();
ds1.setDimensionFilter("COUNTRY", "Australia");
ds1.setRefreshPaused(false);
```

Pause the refresh of invisible charts or tables until they become visible

Pause the refresh of invisible charts or tables until they become visible. This delays the loading of the data of the invisible charts or tables up to the point when they become visible.

Example: (One widget)

In the following example, an analytic application contains a TabStrip widget with two tabs. The second tab contains a table. Whenever the second tab becomes visible, then disable Pause Refresh of this table. Whenever the second tab becomes invisible, then enable Pause Refresh of this table.

1. Enable Pause Refresh for this table in its *Builder* panel section *Properties* > *Pause Data Refresh*.
2. Add the following code to the `onSelect` event script of the TabStrip widget:

```
var selectedTab = TabStrip_1.getSelectedKey();
if (selectedTab === "Tab_2") {
    Table_1.getDataSource().setRefreshPaused(false);
} else {
    Table_1.getDataSource().setRefreshPaused(true);
}
```

Example: (Two widgets)

In the following example, an analytic application contains a TabStrip widget with two tabs. The second tab contains two tables. Whenever the second tab becomes visible, then disable Pause

Refresh of both tables. Whenever the second tab becomes invisible, then enable Pause Refresh of both tables.

1. Enable Pause Refresh for both tables in their *Builder* panel section *Properties > Pause Data Refresh*.

2. Add the following code to the `onSelect` event script of the TabStrip widget:

```
var selectedTab = TabStrip_1.getSelectedKey();
var ds1 = Table_1.getDataSource();
var ds2 = Table_2.getDataSource();
if (selectedTab === "Tab_2") {
    Application.setRefreshPaused([ds1, ds2], false);
} else {
    Application.setRefreshPaused([ds1, ds2], true);
}
```

Pause the refresh of a chart or table, modify it, then update the chart or table

If you, as an end user, need to apply interactively several script operations to modify a chart or table but don't want to wait for the data to be refreshed after each operation, then enable Pause Refresh to disable the data refreshes to the chart or table, apply your interactive operations, and disable Pause refresh. The chart or table refreshes its data, according to your modifications.

Example:

In the following example, an analytic application contains a table, three filter buttons, and a switch. When the switch is set to "on", then Pause Refresh is enabled. When the switch is set to "off", then Pause Refresh is disabled (like in the screenshot below).

The screenshot shows a user interface for an analytic application. At the top, there's a toolbar with 'Edit', 'Tools', 'Display', and other icons. Below the toolbar, the title is 'BestRunJuice_SampleModel' with a subtitle 'in Million USD'. To the right of the title is a toggle switch labeled 'Pause Refresh' which is currently turned on (grayed out). Below the title, there's a table with two columns: 'Account' and 'Gross Margin'. The table has three rows: California (173.48), Nevada (13.25), and Oregon (48.30). To the right of the table are three buttons: 'Set Filter "California"', 'Set Filter "Nevada"', and 'Set Filter "Oregon"'. The entire interface is contained within a rounded rectangular frame.

This allows the end user to apply a filter by clicking a button (or in a more elaborate example, for example, any combination of filters), to redecide, and to apply another filter without waiting for the data to be fetched from the backend. Note that during this phase the data displayed in the table doesn't change. When the end user eventually turns the switch to "off", the data is actually fetched according to the current filter setting and displayed in the table.

You can accomplish this by the following `onChange` event script of the Switch widget:

```
if (Switch_1.isOn()) {
    Table_1.getDataSource().setRefreshPaused(true);
} else {
    Table_1.getDataSource().setRefreshPaused(false);
```

```
}
```

Pause Refresh and Planning

Here are some general rules for using Pause Refresh with planning-enabled tables:

- Enable Pause Refresh with planning-enabled tables when they become invisible at runtime. This avoids the refresh of these tables after any user input to another table that shares the same planning model.
- Disable the planning of a table before enabling Pause Refresh of the table's data source.
- Disable Pause Refresh of a table's data source before enabling the planning of the table.

Example:

You've an analytic application with several planning-enabled tables. You want to leverage the rules above to improve the performance of your analytic application.

1. Disable planning for the tables in their *Builder* panel settings.
2. Enable Pause Refresh for the tables in their *Builder* panel section *Properties > Pause Data Refresh*.
3. Add the following script functions to your application:

```
function activateTables(tables: Table[], activatePlanning: boolean[]): void

for (var i = 0; i < tables.length; i++) {
    var tbl_ds = tables[i].getDataSource();
    var tbl_pl = tables[i].getPlanning();
    if (tbl_ds.isRefreshPaused()) {
        tbl_ds.setRefreshPaused(false);
        if (activatePlanning[i]) {
            tbl_pl.setEnabled(true);
        }
    }
}

function deactivateTables(tables: Table[]): void

for (var i = 0; i < tables.length; i++) {
    var tbl_ds = tables[i].getDataSource();
    var tbl_pl = tables[i].getPlanning();
    if (!tbl_ds.isRefreshPaused()) {
        if (tbl_pl.isEnabled()) {
            tbl_pl.setEnabled(false);
        }
        tbl_ds.setRefreshPaused(true);
    }
}
```

4. When one or more tables become visible at runtime (for example, by switching a tab), then call the function `activateTables()` and pass the relevant tables with the `activatePlanning` flags set to false.

5. When one or more tables become visible at runtime (for example, by switching a tab) **and you want to start planning with them**, then call the function `activateTables()` and pass the relevant tables with the `activatePlanning` flags set to true.
6. When one or more tables become invisible at runtime (for example, by switching a tab), then call the function `deactivateTables()` and pass the relevant tables. If they're planning-enabled, then their planning will be automatically disabled.

Pause Refresh and avoiding unnecessary model refreshes

- Some actions, for example, data action trigger, data input, and member update, cause a model refresh. Use the Pause Refresh API to avoid unnecessary model refreshes.
- Combine the mass data entry mode with Pause Refresh to achieve a better performance.
- Use Data Action components with their script API to trigger multiple data actions, then trigger a data refresh manually with the script API, for example, with
`Table.getDataSource().refreshData();`

Pause each widget individually for a group of widgets

When you want to pause a group of widgets, pause each widget individually.

```
ds1.setRefreshPaused(false);
ds2.setRefreshPaused(false);
ds3.setRefreshPaused(false);
ds4.setRefreshPaused(false);
```

Avoid using `Application.setRefreshPaused([], false)` as it will wait for the queries return before executing the next script.

Avoid this:

```
Application.setRefreshPaused([ds1, ds2, ds3, ds4], false);
```

More Tips

- Disable Pause Refresh before saving bookmarks.

14.5.14 Use Application.refreshData() to refresh multiple data sources

When you want to refresh multiple data sources, use the `Application.refreshData()` API like this:

```
Application.refreshData([ds1, ds2, ds3, ds4]);
```

The script is fully executed without waiting for all the associated widgets to be updated.

Avoid refreshing each widget individually.

```
ds1.refreshData();
ds2.refreshData();
ds3.refreshData();
ds4.refreshData();
```

15 The End and the Future

Dear Reader, we hope you've enjoyed the book. We will enhance the content in the future with the latest features. Now go ahead and have fun building awesome analytic applications!

16 Important Links

Open the SAP Help page to find many more information about SAP Analytics Cloud, analytics designer:

https://help.sap.com/viewer/product/SAP_ANALYTICS_CLOUD/release/en-US

- The official documentation
- The API reference guide
- The SAP Analytics Cloud community
- The SAP Analytics Cloud Wiki
- Many more links

© 2019 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See www.sap.com/copyright for additional trademark information and notices.