Confirmit Horizons v24

This is document revision 1 of the Confirmit Horizons v24 Studio User Guide published in March 2019. The information herein describes Studio and its features as of Build nr. 24.0.820 shown in the **Home > Help > About** box. New features may be introduced into the product after this date. Go to www.confirmit.com or check “News” on the Customer Extranet for the latest updates.

Table of Contents

What's New in this Issue?

Note: Only the latest changes to this documentation are listed here. Changes made to earlier revisions are listed in the "Changes to the User Documentation" document which can be downloaded from the Confirmit Extranet at https://extranet.confirmit.com.

This is the first revision of the Confirmit Horizons v24 Studio User Guide. Both this document and the Studio application are under development, so errors and omissions in this document are to be expected.

Note: The general layout and language in this document is continually being corrected, adjusted and improved to ensure the user has the best possible source of information. Only NEW information and details of functionality that has changed since the previous issue are listed here - minor corrections to the text and document layout are not listed.

Important
We need your feedback so we can improve this document and provide you with the information you require. If you have any comments or constructive criticism concerning the content or layout of this documentation, please send an email to documentation@confirmit.com. Please include in your email the section number and/or heading text of the section to which your comment applies.

1. Introduction

In today's highly competitive business environment, it is more important than ever before for companies to identify and address issues that may have a negative impact on the customer experience, employee engagement, or the wider market's perceptions of the brand.

Confirmit's Studio™ is a platform for designing research solutions for different types of stakeholders, either in the form of full end-to-end workflows from survey creation to reporting and taking action, or in the form of engaging dashboards that help bring the right insights to the right people. Studio helps users to design programs to collect, interact with, explore, visualize and take action based on data collected from research programs, in conjunction with other business systems' data to help businesses answer the burning question "why" and find out what undiscovered issues may be present.

Confirmit Studio™ is an easy-to-use tool that facilitates deep exploratory analysis using innovative analytical techniques.

It enables users to:

- Build workflows that users can follow to create and deploy surveys.
- Build dashboards that enable users to analyze data combined from multiple sources across businesses, and conduct ad-hoc analysis of all data sets from SmartHub.
- Perform automated exploratory analysis and visualization on research programs, in conjunction with other data from internal and external sources.
- Use built-in analysis views that illustrate insights.

The benefits of Confirmit Studio™ include:

- Identifying the deeper insights that impact the business – those that are often hidden beneath the surface.
- Eliminating data silos that hide the most important insights.
- Saving time and effort by employing automated analysis.
- Improving business collaboration and communication and taking action based on insights.
- Displaying analytical findings in a format that is easy for non-analysts to understand.

This manual provides an introduction to the Studio platform. It describes the principles of using the system and gives details of the functionality and properties that are common to multiple solutions created using Studio. Details applicable only to specific solutions are covered in the separate user guides for those particular solutions.

Note: Studio Designer is a company add-on. Contact your account manager if you would like to use Studio.

1.1. Glossary

Below are listed a number of commonly used terms and expressions, with associated explanations:

- **Dashboard Designer / User -** the author or co-author of a Studio report. This user must have Manage permission to the report in order to see and edit the CDL. This user can grant permissions to other users, and publish the report to make it accessible to end-users.
- **Data expression -** a function that can be used to retrieve or calculate data from the hub source.
- **Derived variable -** a variable that you can create by adding the **derived** code to the hub config code block (see Derived Variables on page 49 for more information). This calculates or categorizes previously existing variables.
- **CDL -** Confirmit Design Language - the code language used for creating and configuring the dashboard. CDL is a type of Domain Specific Language (DSL) a computer language specialized to a particular application domain.
- **End user -** a person who ultimately uses or is intended to ultimately use the Studio report or dashboard. Note that this is different from a user, who creates or maintains the report or dashboard - see User above.

- **Filter panel -** the area in the dashboard where all the filters are located. Note that filters can also be included in other locations, for example in widgets and on pages.
- **Fixed filter -** a filter configured to always be applied. It can't be switched off by an end user, but depending on the filter configuration, the end user may be allowed to change it to another value.
- **Formatter -** a block of CDL code that defines how a specific value or property in a widget is to look or behave (see Formatters on page 53 for more information).
- **Intelligent Code Completion** - ICC - help and suggestions provided by the application while you are creating a report, to assist you with achieving correct code.
- **Layout area -** a distinct area of the dashboard, for example the header or the toolbar. A layout area must be defined in the code for it to be visible in the dashboard. If no specific settings are stated then it will adopt the default settings for the area type.
- **LTR -** Likelihood to Recommend - a metric used to quantify how enthusiastic the respondent is regarding a product or service; how likely they are to recommend the product or service to others. This is relevant mainly to VoC users.
- **Metric -** some sort of measurement, for example Average, NPS.
- **View mode -** shows the published version of the report for a user who has been granted View access permission to it. Note that this is the only available mode for an End User.

1.2. What is the Confirmit Design Language?

The Confirmit Design Language (CDL) is a Domain-Specific Language (DSL) used to configure the workflows and dashboards in Studio and the various associated components, for example the data sets in a hub, hierarchies, surveys and contact databases.

The CDL provides you with an efficient way of configuring solutions in Studio, where you can easily reuse a previous configuration by copy/paste, and then making adjustments to the settings. The entire Studio report can be configured with CDL, from overall report settings, configuration of what hub and data sets to use and the relationships between them, pages, widgets on pages, filter panels and more.

Studio provides an editor for efficiently working with the CDL code, with a preview of the report that will continuously refresh as the CDL is updated, various syntax assistance (code completion, syntax highlighting, documentation, validation).

2. Accessing Studio

Note: Studio Designer is a company add-on. Contact your account manager if you would like to use Studio.

To access Studio:

- When you are logged into Confirmit Horizons, click **Studio** from the Home page or the waffle menu at the top of the page.
- When you are in Professional Authoring, go to the **Home > Studio** menu item or click **Studio** in the Quick Access pane.

The first time you open Studio, the Create New Report page opens. After you have created your first report (see Creating a New Report on page 20 for more information) or once you have been given access to existing reports created by other users, the Report List is displayed (see The Reports List on page 6 for more information).

Note: When you access Studio you will only be able to see reports that you have created, and other reports to which you have specifically been given access. Access to reports is controlled via user permissions (see User Permissions on page 3 for more information).

2.1. User Permissions

By default you as the report owner have full control over your report, and initially you are the only person who can view it and edit it. You can allow other Studio users to view and/or edit your report by allocating Professional User access permissions to those users, and you can allow end users to view the published report by setting up an end user list. Each report must have an end user list, though a list can be used for multiple reports.

Note: The end user list must be created in Confirmit Professional Authoring before you can use it to provide end user access to a Studio report. Refer to the Administration of End Users chapter in the Confirmit Professional Authoring User Guide for further details.

2.1.1. Adding Users to the User List

To add users to a report's User List:

1. Open the relevant report and select **Report Settings > User Permissions**.

   The User Permissions page opens.

2. Click the **Add Users** button in the top right corner of the User Permissions page.

   The Add Users overlay opens, displaying the list of all users who are registered in the Report Designer's company and who have been added to the hub on which the report is based.

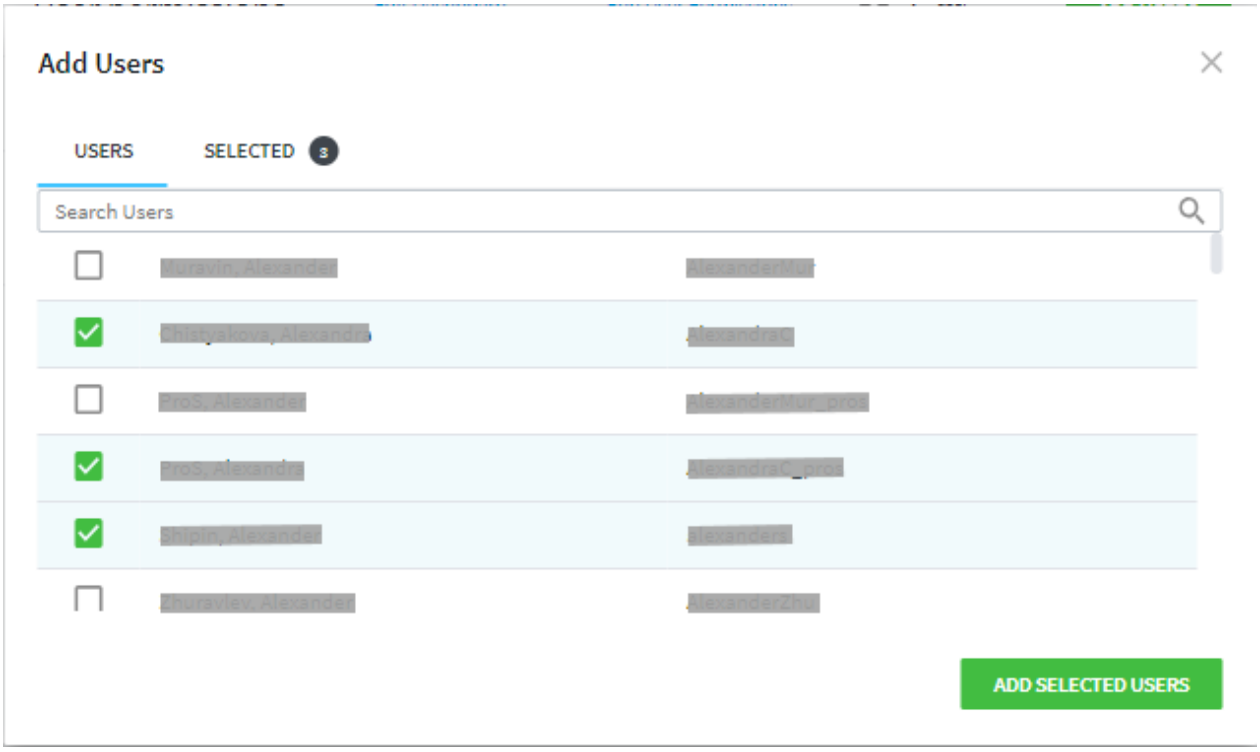


Figure 1 Example of the Add Users overlay

3. Select the users you want to add to the user list by clicking the box in the left column.

   If the list is extensive you can search and filter the list by typing a character or string of characters into the Search Users field. As you type, the user list is reduced to show only those users whose name includes that character or series.

4. When the required user names are selected, click the **Add selected users** button.

   The users are added to the user list.

By default, the newly added users are granted the View access level. You can change this as necessary (see Assigning User Permissions to a Report on page 4 for more information).

2.1.2. Assigning User Permissions to a Report

To set the access level for a Studio user:

1. Open the report and select Edit mode, then click the **Report Settings** icon .
2. From the drop-down menu select **User Permissions**.

   The User Permissions page opens. This page lists all the Studio users who currently have access to the report. The current level of access for each user is indicated in the Project access column.

3. Click in the Project access field for the required user, and select the access level you wish to give to that user from the drop-down list .

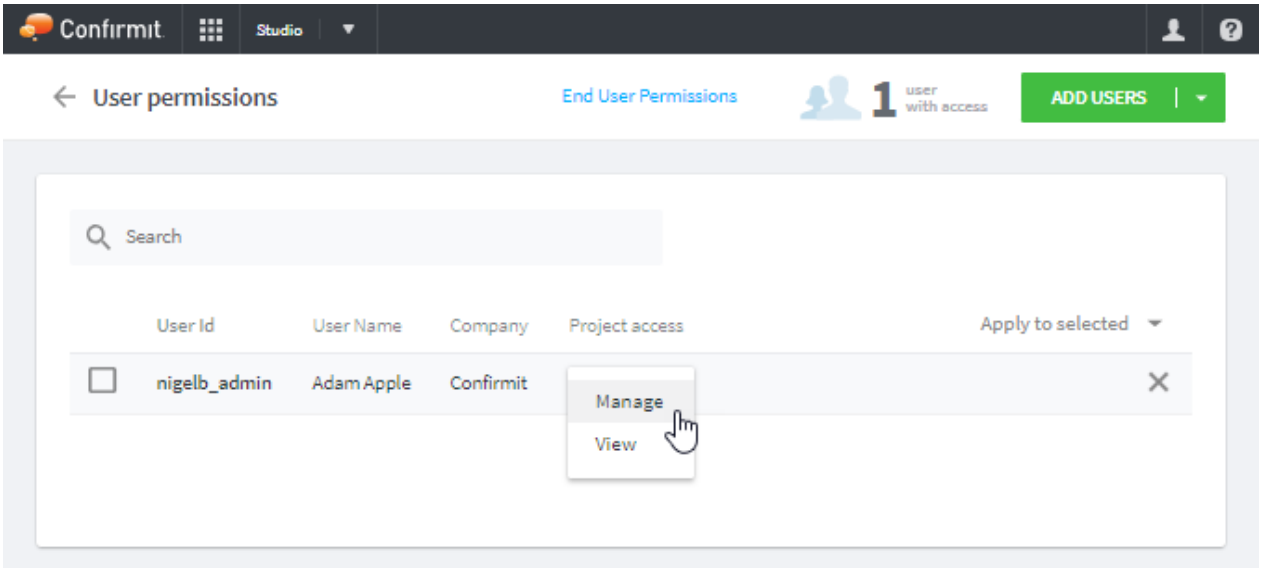


Figure 2 Setting the Manage report access level for a user

The options are:

- **Manage -** the user has full access to the report and can edit or delete the report as necessary.
- **View -** the user can only view the report and its CDL; he/she cannot edit it.

To revoke access to the report, click the **Revoke Access** button beside the Project access field.

When you have a number of users added, to apply a permission or revoke access for several users simultaneously, check the appropriate boxes in the left column to select the users, then use the **Apply to selected** drop-down menu. This menu contains the following options:

- **Assign Manage permission** - the users can edit the report as required.
- **Assign View permission** - the users can only view the report and its CDL.
- **Revoke Dashboard access** - the users are revoked access to the report.

To apply a permission or revoke access for all users simultaneously, check the **Select All** box in the left column header to select all users, then apply the appropriate option in the **Apply to selected** drop-down menu.

When you are finished editing the user permissions, click the left-arrow beside the page name to return to the project.

2.2. End User Access

End user access to a report is managed via an end user list. The end user list must be created in Confirmit Professional Authoring, and it must be linked to the same hub that is providing data to the Studio report.

All end users in the list will initially have the same level of access; full access to the entire report. However it is possible to define roles for the end users, for example in a custom data table that links userid to a role. The roles can then be used to control what the end user has access to in terms of content (pages) and data (filters) (see Role Based Access on page 73 for more information).

Refer to the Administration of End Users chapter in the Confirmit Professional Authoring User Guide for further details.

2.2.1. Assigning an End User List to a Report

Once you have the appropriate end user list it must be assigned to the relevant report. To do this:

1. When in the report, click the **Report Settings** icon .
2. From the drop-down menu select **End User Permissions**.

The End User Permissions page opens. This page lists all the end user lists currently assigned to the report.

3. Type the ID of the end user list that you wish to assigned to the report into the End User List ID field.
4. Click the **Add User List** button.

   The list ID and name are displayed in the Assigned End User List.

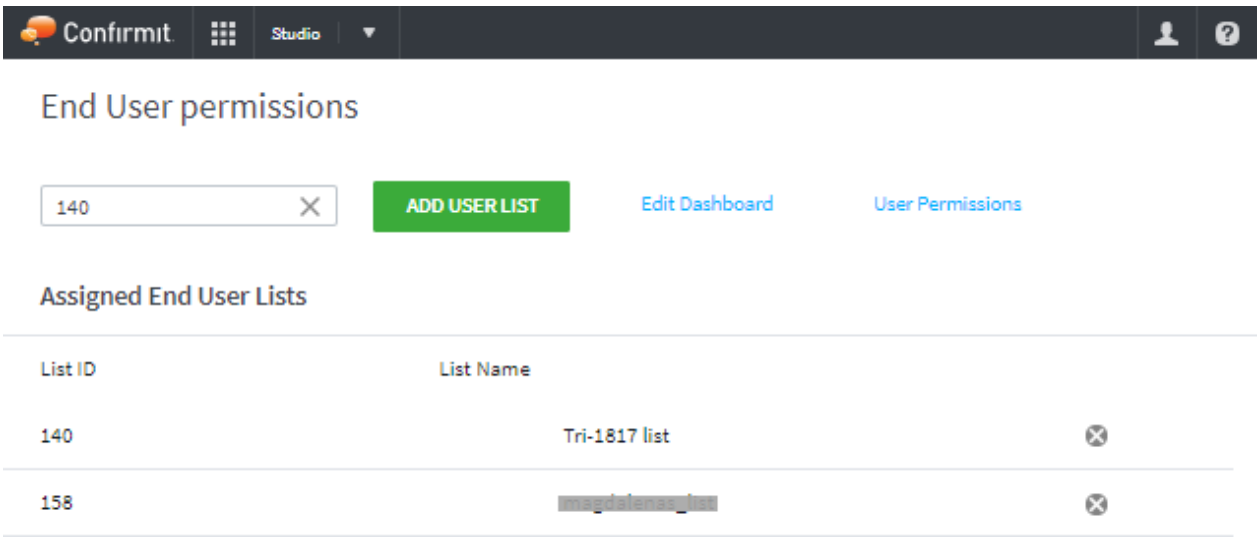


Figure 3 The End User permissions page with the assigned End User lists

Note: You can assign multiple end user lists to a report.

2.3. The Reports List

The Studio Reports page displays the reports you have created, and other reports that you have specifically been given access to by the report owner (see Assigning User Permissions to a Report on page 4 for more information). The Studio Reports page contains two tabs:

- **Recently Accessed -** lists the reports you have recently been working with.
- **All Reports -** lists all the reports to which you have access.

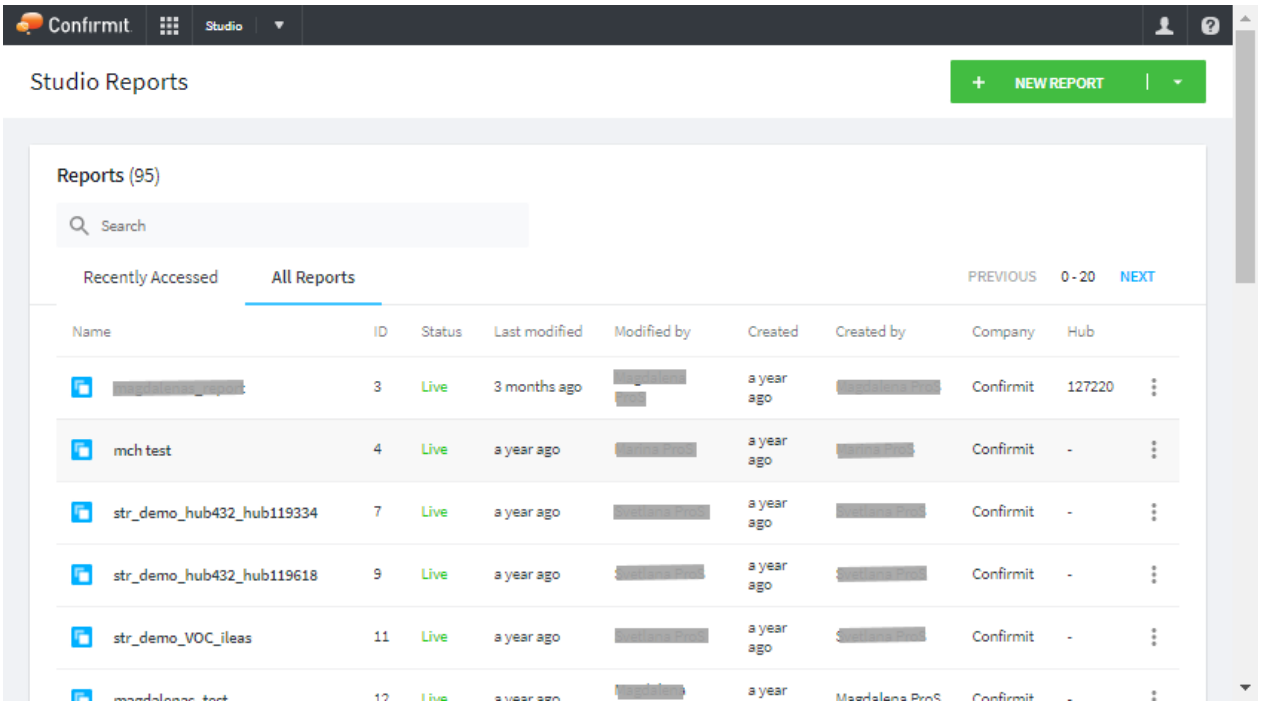


Figure 4 The Reports list

The Reports list presents the reports with 20 reports to a page, in the following columns:

- **Name** - the report name as assigned by the report creator (see Creating a New Report on page 20 for more information).
- **ID** - the report ID. This is a number assigned automatically by Studio to the report, in order of creation, for the installation.
- **Status** - the report status. Until the Report Designer has published the report, its status is displayed as **Draft**; after it has been published, the status is changed to **Live**.
- **Last modified** - when the report was modified (in relative format).
- **Modified by** - the name of user who last modified the report.
- **Created** - when the report was created (in relative format).
- **Created by** - the name of user who created the report.
- **Company** - the name of the company to which the creator is registered.
- **Hub** - the ID of the Hub that provides the data used in the report.
- For each report, click the **Report management** icon ⋮ to open the report management menu. This contains the following commands:
    - o **Edit** - opens the current Draft version (see Publishing a Report on page 29 for more information) of the report in Edit mode with the CDL Editor panel (see Defining a Report on page 21 for more information).
    - o **View** - opens the current published version of the report in View mode (see Previewing a Report on page 29 for more information)
    - o **Delete** - allows you to delete a report that is no longer required (see Deleting a Report on page 31 for more information)

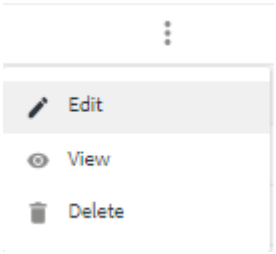


Figure 5 The Report Management menu

To create a new report, click the **New Report** button in the upper-right corner of the page (see Creating a New Report on page 20 for more information)

The **PREVIOUS** and **NEXT** buttons towards the right side of the list's toolbar allow to you page through the Reports list.

In the event the list is extensive, the **Search** input field allows you to search the report list (see Searching the Report List on page 8 for more information)

2.3.1. Searching the Report List

In the event your report list is extensive, to more easily find the report you wish to work with you can search the list. To do this, type a character or string of characters into the **Search** field. As you type, Studio filters the report list to show only those reports that include that character or string of characters in one (or more) of its Report List columns.

To clear the search criteria, click the **X** icon beside the Search field.

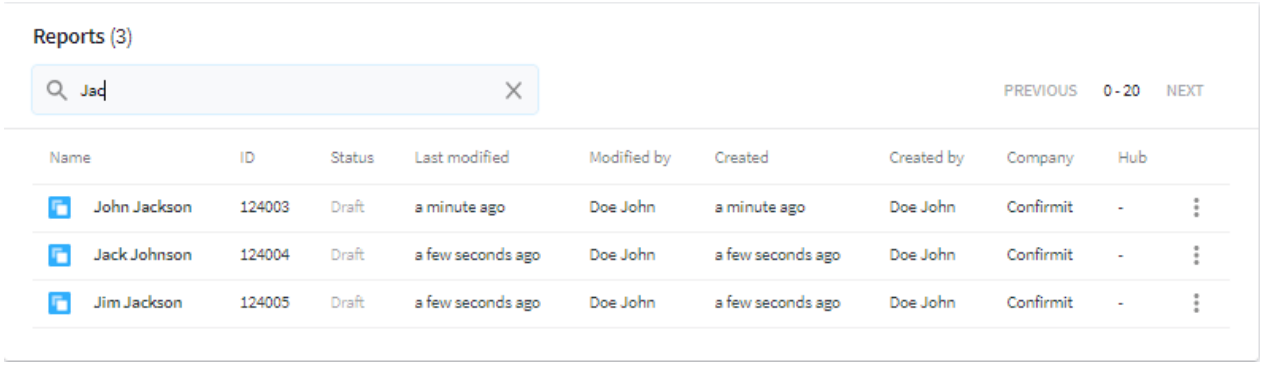


Figure 6 Using the Report List search functionality

3. The Report Editor Page

When you click on a report name in the Report list, the report opens in the Report Editor page. This page is divided into two main areas; the CDL Editor Panel on the left where the solution is defined (see The CDL Editor panel on page 10 for more information), and the Preview Panel (may be called the canvas) on the right (see The Preview Panel or Canvas on page 19 for more information).

Note: To reduce the chances of changes being made accidentally, a report will always open in Read Only mode. Click the Edit button in the toolbar or the Edit icon in the blue Read Only mode bar to open the report for editing. While the report is open for editing the button in the toolbar reads Done Editing - click this to return to the Read Only mode.

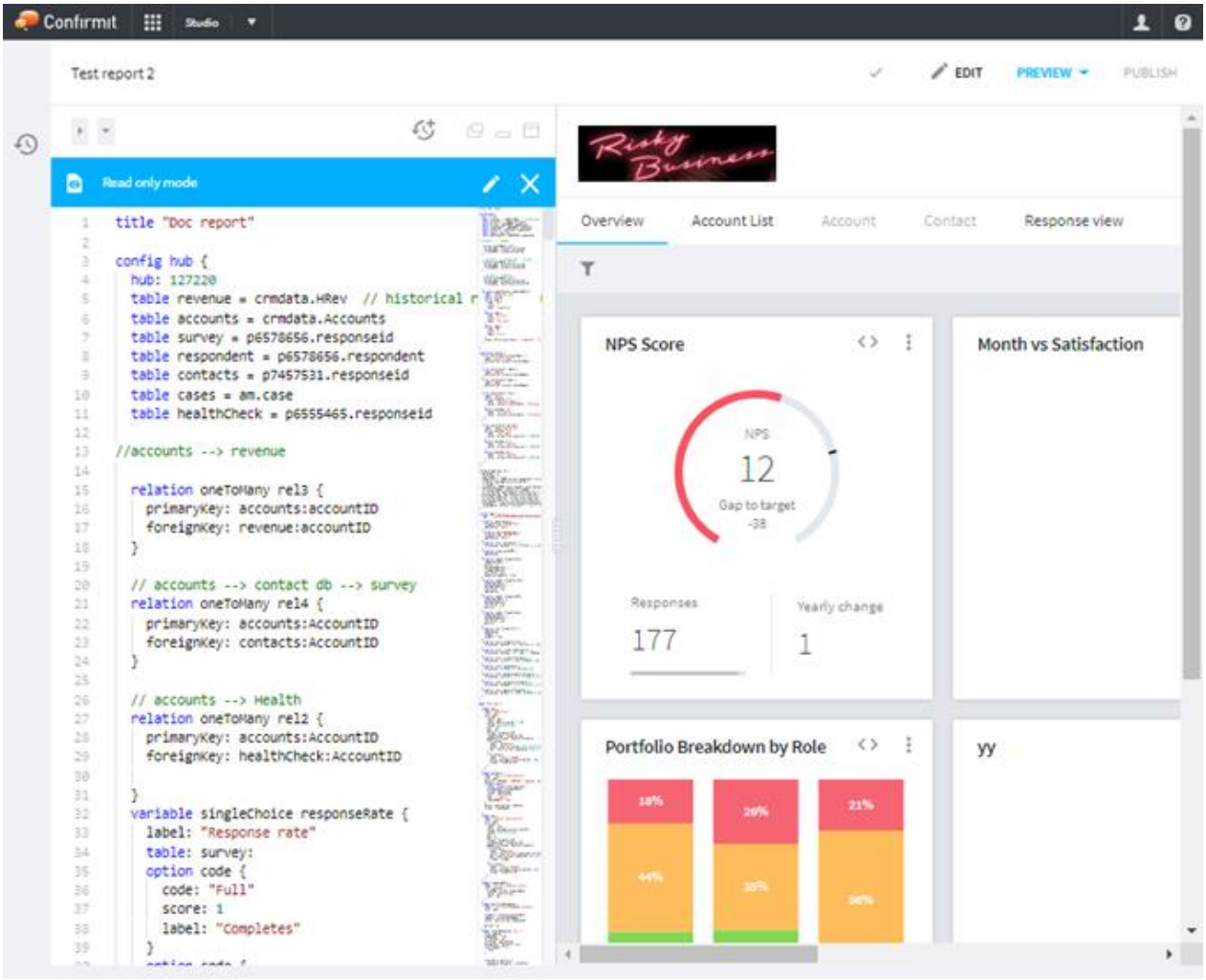


Figure 7 The Report Editor page in Read Only mode

The fields, tools and icons in the Studio toolbar are:

- **Report Name -** presents the name of the report (see Creating a New Report on page 20 for more information). You can edit this if necessary; click into the Report Name text to open the field and change the text as required. Note that you do not change the report name by changing the title code in the CDL Editor panel.

- **Saved -** in Draft mode, any changes you make to the report will be saved after a few seconds, while revisions must be saved manually. This indicates the current status of those changes (see Saving a Report on page 25 for more information).
- **Report Settings icon -** opens a menu enabling you to provide access to the report to other users and end users (see User Permissions on page 3 for more information). Click outside the menu to close it.
- **Done editing -** while the report is open for editing the **Done Editing** button is visible in the toolbar - click this to return to the Read Only mode.
- **Preview -** (see Previewing a Report on page 29 for more information)
- **Publish -** click to publish the report (see Publishing a Report on page 29 for more information). This will make it available to the end users who have been given access (see End User Access on page 5 for more information).

3.1. The CDL Editor panel

The CDL (Confirmit Design Language) Editor panel is the area in the left part of the Report Editor page where the Report Designer builds and defines the report (see Defining a Report on page 21 for more information).

The Editor panel is divided in two areas.

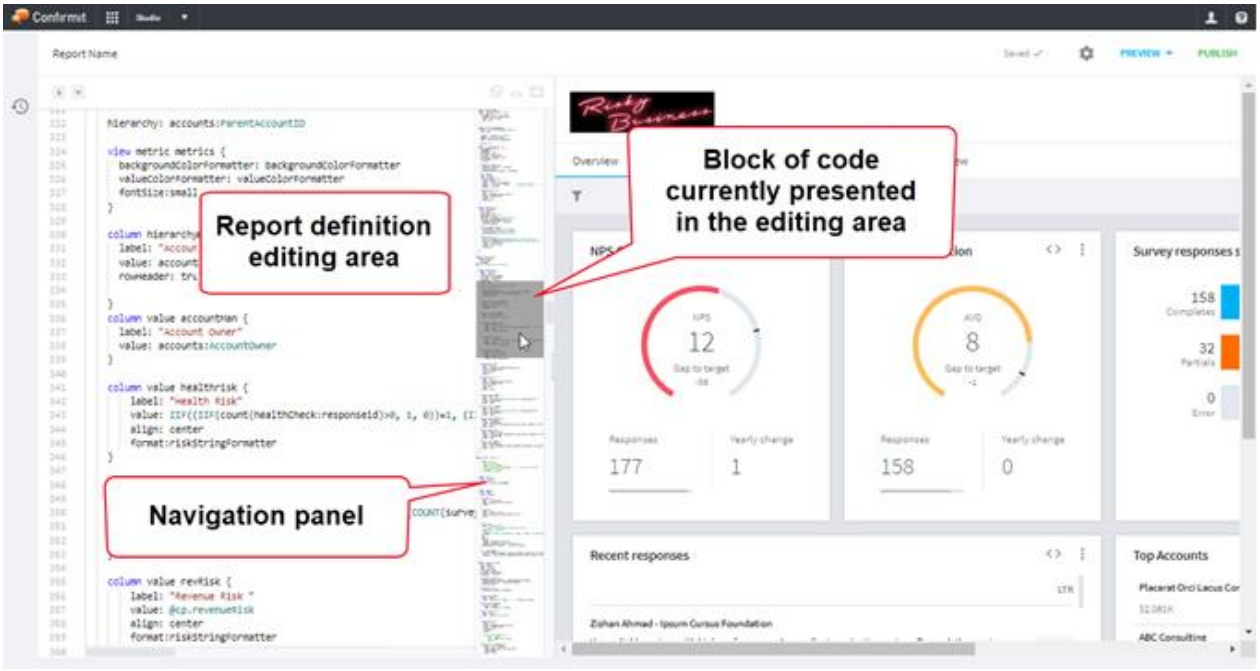


Figure 8 The CDL Editor panel

The wider pane to the left is the report definition editing area where you actually build the report. The font used to present the code is easily readable, and the different code types and entities are color-coded to simplify identification.

The narrow panel to the right is the navigation panel. This shows a compressed block of the code to enable you to quickly and easily navigate through the report definition. Hover your mouse pointer in the navigation pane to highlight the actual block of code that is currently presented in the editing area, and scroll or drag the highlighted area to move to the block of code you wish to view.

The toolbar across the top of the editor panel contains tools allowing you to manipulate the panel.

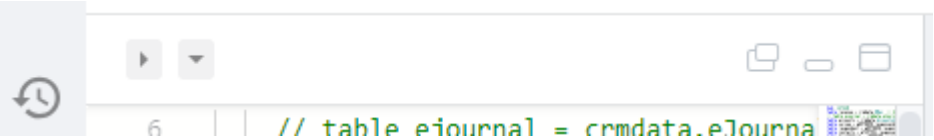


Figure 9 The CDL Editor panel toolbar

- collapses all the nodes in the editing panel.
- expands all the nodes in the editing panel after they have been collapsed.
- detaches the editing panel from the Studio window allowing you to move the panel to another location in your screen. Note that this is still under development.
- minimizes the CDL Editor panel, leaving the entire Studio window to display the preview. Click the **Editor** tab to re-open the panel.
- maximizes the editor panel so it fills the entire Studio window. The larger screen may simplify your report building work as you will have more space to work in. Click the **Dashboard** tab to reduce the editor panel to normal size and redisplay the preview panel.

Note that you can re-size the editing panel - drag the right edge of the panel to the desired width.

Tip
Press **Ctrl+z** on your keyboard to undo your last change. You can repeat the action to undo earlier changes back to the start of your current session.

3.1.1. Intelligent Code Completion

The CDL Editor uses Intelligent Code Completion (ICC) functionality to help you to build the right syntax by providing suggestions as you type into the Editor. In the example below, the user has typed **con** into the editor and a list of all the entities that include the characters **c**, **o** and **n** in any order is presented.

Note: To invoke code completion manually, press the Ctrl+Space keys on your keyboard.

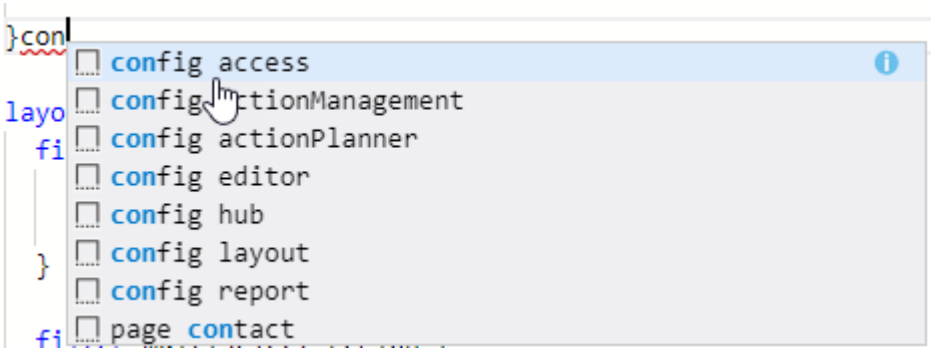


Figure 10 Example of a suggestion list

A maximum of 12 options are presented in the list, though many more may be available. Move your mouse pointer into the list overlay to show a scroll bar, then drag the scroll bar slider or use your mouse roller to find the desired item, then click on it to select it. Or use the **Up** and **Down** arrow keys on your keyboard to move the highlight through the list, and press **Return** or click on the item to select it. Note that if you click on an entity it will be selected.

Click on the **Info** icon for an entity or press the **Ctrl+Space** keys on your keyboard to open a pop-up providing additional information for that entity.

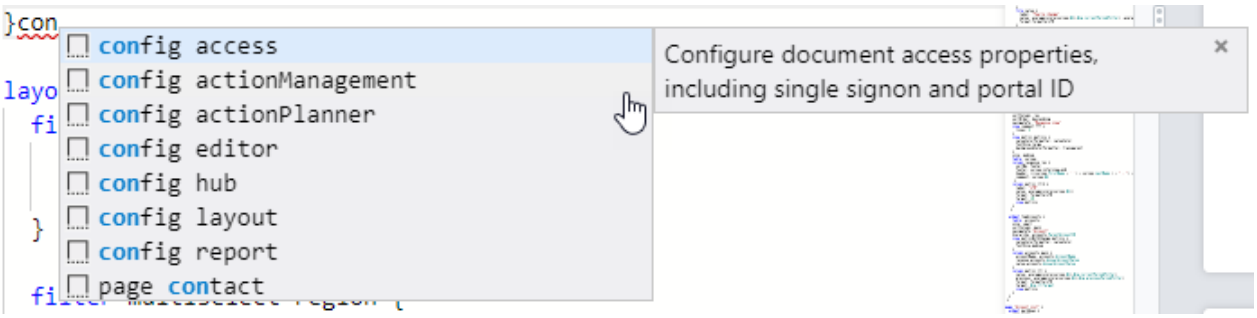


Figure 11 Example of the additional information text for an entity

To select the desired entity, once it is highlighted click on it with the mouse or press the **Tab** or **Enter** keys. When you select an entity it is automatically created in full in the editor, along with any brackets etc. to ensure the entity is "valid" for the compiler.

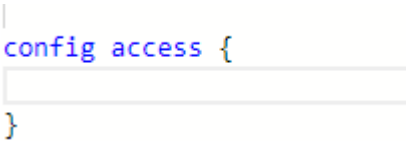


Figure 12 The entity with associated brackets is created

Important
For the ICC to open a list of suggestions, you must start to type a "recognized" character string at an "acceptable" location in the editor panel. If your cursor is at a location in the editor panel where the entity you want to create is not allowed, or if the character string you type is not the first few characters of a valid entity name, then a suggestion list will not appear.

3.1.2. The CDL Editor Context Menu

When you right-click anywhere inside the CDL Editor pane (see The CDL Editor panel on page 10 for more information), the context menu is displayed:

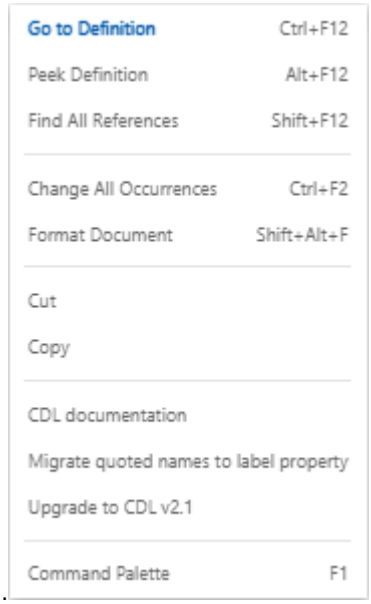


Figure 13 The CDL Editor context menu

- **Go to definition [Ctrl+F12] –** if the code is referencing something that is defined elsewhere, this takes you to the place in the code where that definition is located.
- **Peek definition [Altl+F12]** – same as above, but this allows you to see the definition without jumping to that place in the code
- **Find all references [Shift+F12]** – it can be useful to understand the impact a change may have by seeing how many times it has been referenced.
- **Change all occurrences [Ctrl+F2]** – finds all the occurrences of the given text in the code and changes them.
- **Format document [Shift+Alt+F]** - re-formats the code with indentation to make the code more readable.
- **Cut -** cuts the selected code from the editor.
- **Copy -** copies the selected code to your clipboard so you can paste it into another location in the editor.
- **CDL documentation** - displays the context help text (see The CDL Documentation on page 13 for more information).
- **Migrate quoted names... -** click to migrate "older format" names using quote symbols, to the newer # symbol. For example **page "Top Account"** to **page #topAccount**.
- **Upgrade to CDL v2.1 -** click to upgrade your report/dashboard to the latest version of the CDL.
- **Command Palette [F1]** – gives an overview of all operations in the editor, with keyboard shortcuts.

3.1.3. The CDL Documentation

The CDL Editor panel provides you with documentation and help pop-up texts to assist you with building and configuring your report. To open the CDL documentation for a particular parameter:

1. Right-click on the parameter you want information for to open the context menu, then select CDL Documentation from the menu.

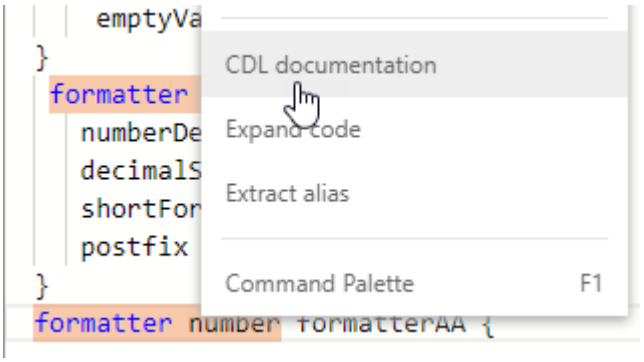


Figure 14 Opening the CDL documentation

The documentation available for that parameter opens.

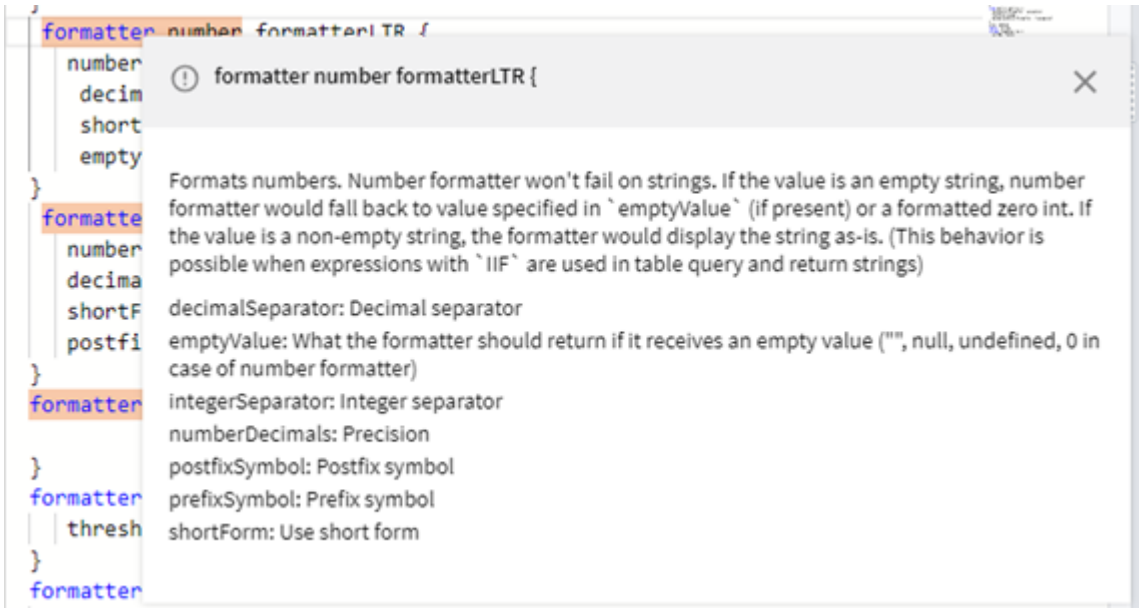


Figure 15 Example of the information available for a number formatter

Here are listed the properties that you can add to the formatter, with a short description of each.

3.1.4. Searching in the CDL Editor Panel

You can search for character strings in the CDL Editor panel. To do this:

1. Click into the CDL Editor panel and press **Ctrl+F** on your keyboard.

   The Find field appears at the top of the editor panel.

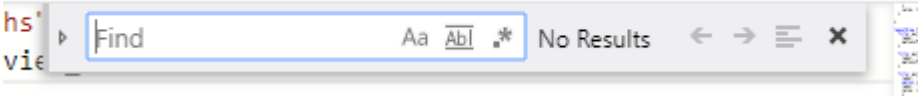


Figure 16 The Find field

2. Type the character string you are looking for into the Find field.

   As you type, Studio moves to the first instance of the string that it finds, and highlights it,

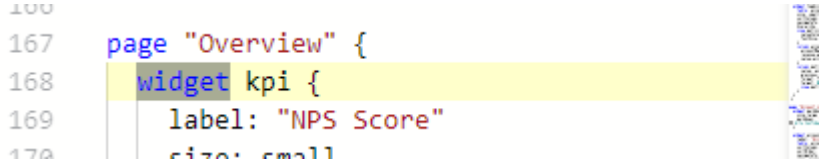


Figure 17 The Find functionality has highlighted the first hit

   and also indicates how many instances of the string exist in the editor panel.

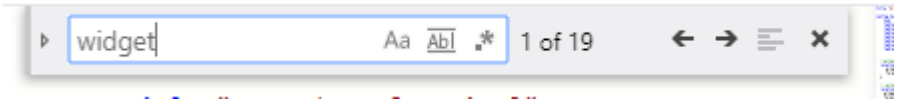


Figure 18 The number of hits are indicated

Click the **Previous match** and **Next match** buttons (or press the **Shift+F3** or **F3** keys on your keyboard respectively) to move between the instances, and when you are finished click the **X** button to close the field.

Click the arrow button to the left of the Find field to open the Replace field, allowing you to perform "find and replace" operations in the code.

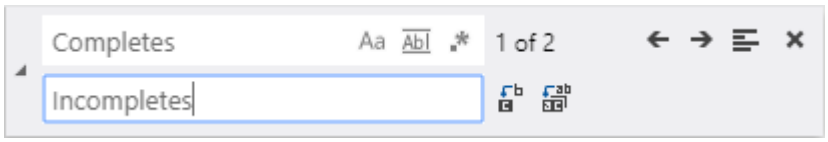


Figure 19 Finding and replacing

In this example, two instances of "Completes" can be replaced by "Incompletes". The tools to the right of the fields have tool-tips to clarify their functions.

3.1.5. The Command Palette

The command palette is a complete list of all the commands you can use in Studio, along with the keyboard shortcuts for the commands where shortcuts are available.

Figure 20 The Command palette

To open the Command palette:

1. Either right-click in the CDL Editor panel to open the Context menu, then select **Command palette** from the menu.

   or

- While your mouse cursor is in the CDL Editor panel, press the **F1** key on your keyboard.

To use a command from the Command palette:

2. Select an area of code or place the mouse cursor as appropriate in the CDL Editor.
3. Open the Command palette.
4. Scroll to the command you wish to use, and click on it to select it.

3.1.6. Comments in CDL

Important
You are strongly recommended to add comments into the CDL at every opportunity.

Comments in the code will greatly simplify building the dashboard, and will also help to pass on the knowledge of those who built/edited the dashboard to those who will inherit it later. This particularly applies to any custom CDL script, precisely because it is custom!

Comments allow you to add descriptions of what you have done and why, and allow you to keep track of what the different blocks in the code are intended to do. You can also use comments to cause the compiler to ignore lines of code. This will for example allow you to hide particular objects while you are building the report until you are ready to include them, and will also allow you to avoid the irritating "Compiler errors" that will appear before the object is complete.

To create a single line comment in CDL, place two slash "//" characters in front of the code or text you wish to be ignored by the compiler. While the two // characters are in place, all text to the right of them on the same line will be ignored. If you want to comment out a block of code, add the two // characters to the start of each line in the block.

```
widget contactSurveys {
  label: "Surveys"
  table: survey:
  // sortColumn: contactID
  // sortOrder: descending
  size: large
  navigateTo: "Response view"
```

Figure 21 Example of comments being used to add useful information to a widget

There may come a time when you need to convert a larger block of code to "comments". For example, perhaps you are working on a widget and you don't want the "in progress" code to disturb the rest of the report while you are developing it. In this case you don't need to comment each line individually; you can select the entire block of code - as much as required - then press the **Ctrl+/** keys on your keyboard. This will add the // characters to every selected line of code simultaneously. To convert the comments back into "normal" code, select the block and press **Ctrl+/** again.

3.1.7. The Color Picker

In objects and properties where you can define colors, for example in color palettes, color formatters etc. if you know the numerical values of the color you want to use then you can merely type these directly into the CDL Editor panel. However a color picker is available that allows you to select colors from a palette using sliders for color, hue and opacity.

To open the color picker from an existing color code, hover the mouse pointer over the code for a couple of seconds and the picker overlay opens. If you are adding a new color to your palette, type the character string **"#000000"** (or any other 6-character hex code) into the appropriate location and then hover over the code. Again, the color picker opens.

Figure 22 Example of the color picker using hex code

You can now select the color you wish to use.

- In the column on the right, drag the vertical slider (black arrow in the figure above) or click on one of the basic colors.
- In the main area, drag the circular slider (white arrow) or click in the area to select the hue you want to use.
- Once you have the basic color, you can then drag the vertical slider (blue arrow) to adjust the opacity if necessary.

A sample of the selected color is presented towards the top of the picker window, along with the numerical values for that color. If you are changing an existing color then a patch of the old color is kept at the right end of the color bar until you move the pointer out of the color picker so you can compare the old and the new colors. A sample patch of the color is also displayed beside the numerical code in the CDL Editor panel.

Note that if you adjust the opacity then the color code changes from a 6-digit hex code to an rgba code.

Figure 23 Example of the color picker using rgba code

Note: If you have multiple color codes in a palette or a color formatter, then the codes must be separated by "," (comma). The last color code is not followed by a comma.

3.1.8. Reducing the Volume of Visible Code

To simplify editing, you can reduce the volume of code that is displayed in the editing area by collapsing blocks of code.

Note that for the code blocks to collapse correctly, the code must first be indented correctly. To do this, right-click in the CDL Editor and select **Format Document** from the drop-down menu or press the **Shift+Alt+F** keys on your keyboard (see The CDL Editor Context Menu on page 12 for more information).

To collapse a code block:

1. Move the cursor into the line number column beside the code block you wish to collapse.

   A **Collapse code** icon ⊟ appears for each code block.

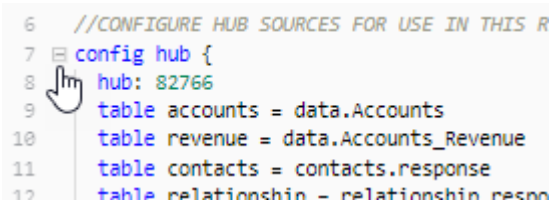


Figure 24 Showing the Collapse Code icons

2. Click the icon to collapse the code to a single line

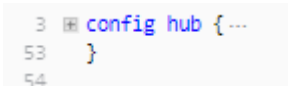


Figure 25 The code block collapsed

Note that the associated line numbering is also collapsed; lines of code after the collapsed block retain their original line numbers. The **Collapse** icon changes to a permanently visible **Expand** icon, and an ellipses character is placed at the end of the collapsed line to indicate that code is hidden.

3. To expand the code block, click the **Expand** icon.

3.1.9. Compiler Errors

A compiler error occurs when the DSL syntax is incorrect, for example a required field is missing or there is a typing error in the code. The compiler for Studio reports "knows" what to expect inside each block of the code, so if it encounters something unexpected it will notify you with an error message presented at the bottom of the CDL Editor panel. Click the up-arrow in the message to open the message. Information about the error is then presented, along with a useful **Go to** link to take you directly to the error. The error message will normally contain enough information so you can understand the reason for the error, and how to rectify it (see Errors and Troubleshooting on page 108 for more information).

3.2. The Preview Panel or Canvas

The Preview Panel (maybe called the Canvas) allows you to preview the report. Click **Preview** towards the right end of the Report toolbar, then select whether you wish to se the draft version or the live version.

4. Working with Reports

When you are creating a report, to ensure the correct items are available when they are needed, the main operations should be performed in a specific order. The general procedure is as follows:

1. Prepare all necessary data sets (see Data Sets on page 22 for more information).
2. Create an empty Report (see Creating a New Report on page 20 for more information).
3. Define the Report in the Report Editor (see Defining a Report on page 21 for more information).

    o Here you should first configure the hub that is to be used, then configure the report itself. You should add the pages on which the widgets are to be presented, then add the desired widgets into the appropriate pages.

4. Grant access to end users by assigning an end user list to the report (see Assigning an End User List to a Report on page 5 for more information).
5. Publish the report to make it available to end users (see Publishing a Report on page 29 for more information).

4.1. Creating a New Report

To create a new report:

1. Click the **New Report** button 

in the top right corner of the Report List page. The New Report overlay will be displayed.

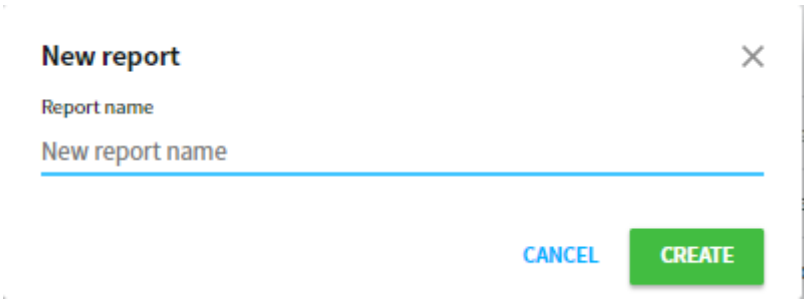


Figure 26 The New Report overlay

2. Click into the New report name field and type in a name for your report. Note that you can change this later (see The Report Editor Page on page 9 for more information).
3. Click **Create** to create the report, or **Cancel** to cancel and close the overlay.

    The Report Page is displayed with the report name as its title. The CDL Editor area on the left contains the report title definition code: `title: "Report_name"`. The Canvas area on the right will be empty.

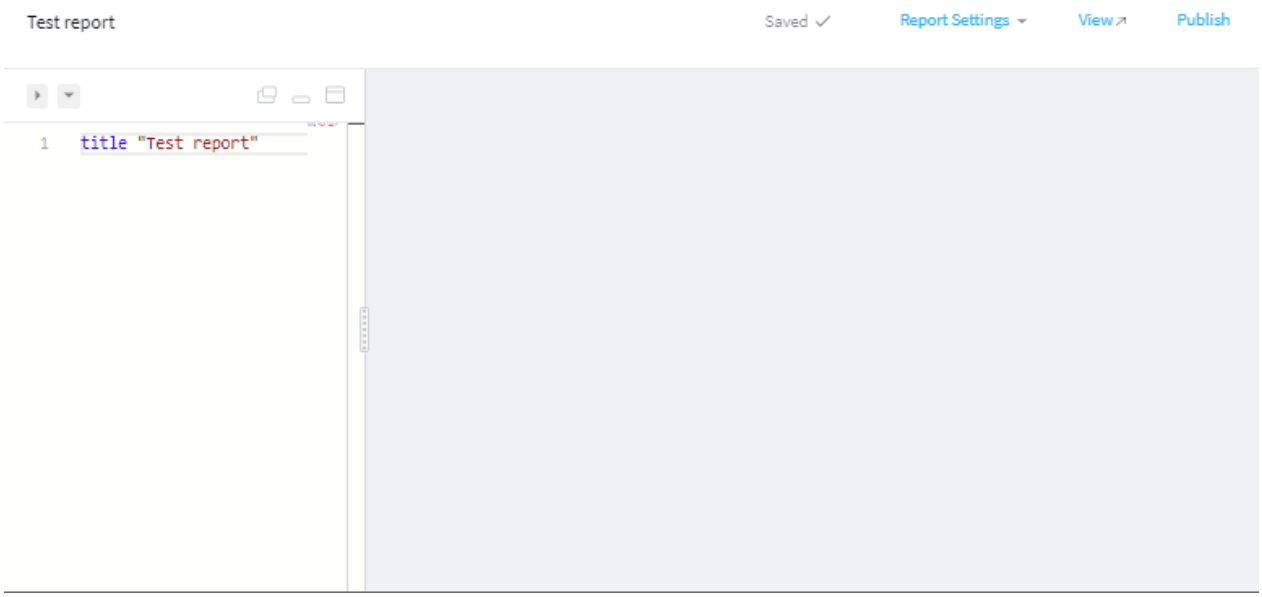


Figure 27 Example of a new, empty report

You now need to define the report. You do this by adding code to the CDL Editor pane (see Defining a Report on page 21 for more information).

The Information pane in the lower left part of the CDL Editor displays any compiler and runtime error and warning messages that may arise (see Errors and Troubleshooting on page 108 for more information).

4.2. Defining a Report

Reports are defined using the Confirmit Design Language (CDL) (see What is the Confirmit Design Language? on page 2 for more information) in the CDL Editor panel (see The CDL Editor panel on page 10 for more information). After you have created a new empty report (see Creating a New Report on page 20 for more information), you build the report definition in the CDL Editor panel, and as you build it you can see the report grow in the Preview panel (see The Preview Panel or Canvas on page 19 for more information).

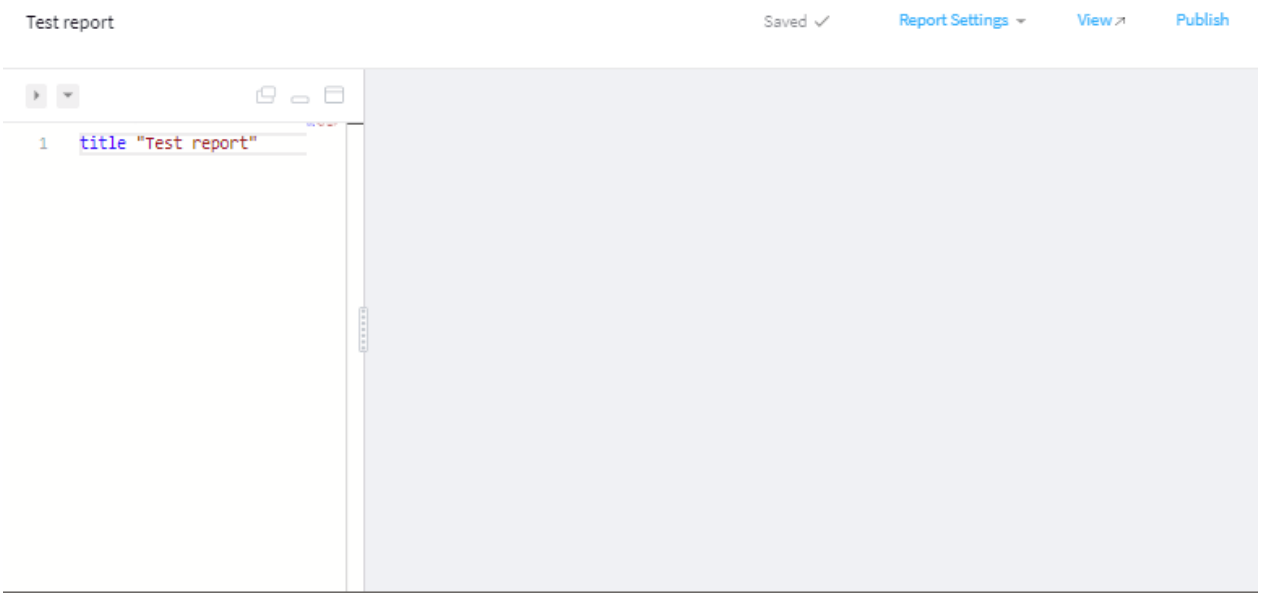


Figure 28 Example of a new empty report

The report definition is organized in functional blocks (see Report Definition Structure on page 45 for more information).

Important
You are strongly recommended to add comments into the CDL at every opportunity (see Comments in CDL on page 16 for more information).

4.3. Data Sets

The data presented by a Studio report is provided by SmartHub. A hub may contain multiple data sets, each of them providing data for corresponding widgets in the report, or it may contain just one data set that contains all the necessary data. A data set may be a survey, a combined survey (including one or more surveys), and other data depending on which metrics are to be monitored and analyzed. The following are some examples:

- Surveys (Relationship, transactional, employee, product, ad tests etc.).
- Contact databases (customer contacts, employees, other).
- Panel databases.
- Case data from Action Management.
- CRM data (Accounts, products etc.).
- Financial data.
- HRIS (Human resources).
- Benchmarks, targets, historic results.

4.3.1. Configuring the Hub

When setting up the report, you must first specify the hub you're going to use then list the tables that are to provide the data, for example survey, contacts and accounts. In the report configuration, hubs are referenced by the hub ID. To find this, log in to Horizons and on the Home page click the SmartHub tile then find the hub you wish to use.

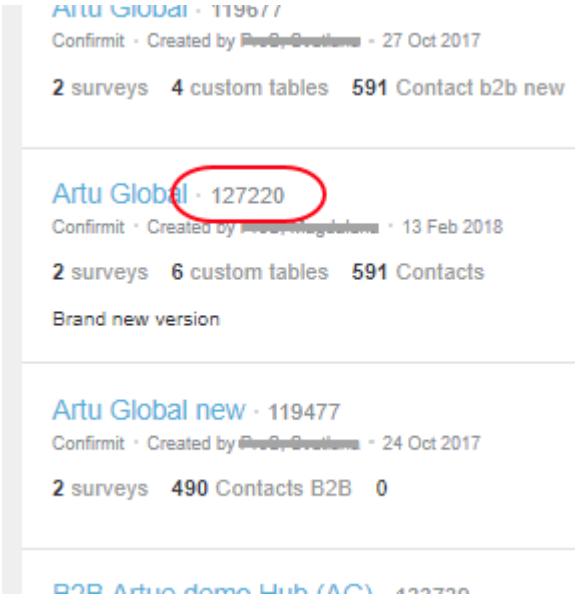


Figure 29 Finding the hub ID in SmartHub

Make a note of the ID, then when you are configuring the hub in the Studio report, add this ID number .

```
3  config hub {
4      hub: 127220
5      table revenue = crmdata.HRev  // historical revenue custom table.
6      table accounts = crmdata.Accounts
7      table survey = p6578656.response
8      table respondent = p6578656.respondent
9      table contacts = p7457531.response
10     table healthCheck = p6555465.response
11     table cases = am.case
```

Figure 30 Example of the hub configuration code block

The you can list the tables that are to be used and give them aliases to make them easier to remember and access. Once an alias is specified, you can then access the table by the name instead of having to type out the full name every time (see Table Aliases on page 48 for more information).

For example, the table specified in row 5 - **cdmdata.HRev // historical revenue custom table** - is here given the alias **revenue**. So from now on in this report, whenever you want to use the data from the cdmdata.HRev // historical revenue custom table, you only need to use the name **revenue**.

Important
You are strongly recommended to add comments into the CDL at every opportunity (see Comments in CDL on page 16 for more information).

4.3.2. Table Relations in a Studio Report

Having defined the hub and listed the tables, you now need to define how the data is to be linked across the different tables.

The relationship between the tables can be one of two types:

- One to many, meaning for each row in one table there can be many corresponding rows in the other table. For example the relationship between the accounts table and the contacts table, where each account can have multiple contacts.

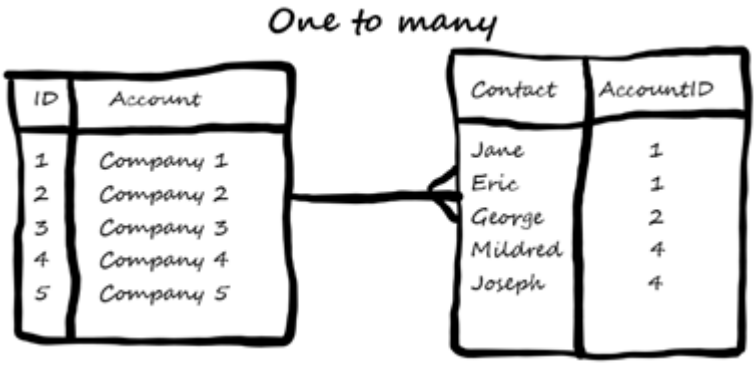


Figure 31 The one-to-many relationship

- One to one, meaning for each row in one table there can be only one corresponding row in the other table. This could be for example that you want to extend one table with information from another table.

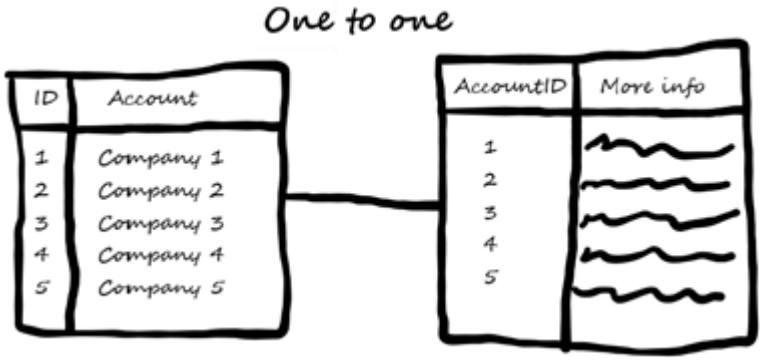


Figure 32 The one-to-one relationship

If the contact database was extracted from the survey, it will already be linked to the survey in a one-to-many relation (one contact, (potentially) many survey responses) behind the scenes, so we don't have to tell Studio about it. If the contact database is not linked to the survey, a separate relation block will be needed. This relation block will tell Studio widgets how to "find" the path to the data when it is located in several different tables. In this case the necessary code will look something like that given below:

```
// relation relationType uniqueIdentifier {
// primaryKey: table:column // column that has unique values for each
row, usually something with ID in name, e.g. AccountID in accounts table
// foreignKey: otherTable:correspondingColumn // column that allows to
cross reference data between two tables
// }
// e.g.
relation oneToMany rel1 {
primaryKey: accounts:Id
foreignKey: contacts:AccountID
}
```

Example code:

Assuming you have following data in your hub:

- A relationship survey (p123456 database, response table).
- A contact database extracted from survey (p123457 database, response table).
- An Accounts table, uploaded as custom data (for example crmdata dataset, accounts table, note that names are specified by the user) (see Custom Data on page 25 for more information).

Then the hub configuration in the Studio report could look as below:

```
config hub {
hub: 1 // hub id
table survey=p123456.response // all the responses are stored in response
table
table contacts=p123457.response // similar as in survey, contacts are
stored in response table
table accounts=crmdata.accounts // names for tables are specified when
uploading custom data, so it's best to use Intelligent Code Completion
(Ctrl+Space) or check the name in the hub
// accounts -> contacts -> survey responses
// contact database is linked to a survey so we don't need to specify the
relation
relation oneToMany rel1 {
primaryKey: accounts:Id
foreignKey: contacts:AccountID
}
}
```

4.3.3. Custom Data

The Custom Data functionality enables you to import any kind of external non-survey data into SmartHub, which we can then reference from Studio and display in a widget. This data can include current and historical revenue numbers, support ticket data, essentially anything that's not stored in a survey response or contact database. A Custom Data table can even include information about child / parent relationships since you can have hierarchical relationships between entities, for example many child accounts under a parent account.

The custom data is stored as and uploaded to SmartHub as Custom Data tables .

The general procedure for using custom data tables is as follows:

1. Set up tables in your SmartHub to bring in non-survey data (accounts, contacts, products, historical revenue, stats on support tickets, etc.).
2. Establish relations between these tables to preserve data integrity.
3. Simplify the creation of a schema for these tables (with variable tables, properties, labels etc.) via an intelligent data types identification, by parsing the data in the file.

For detailed information refer to the separate SmartHub User Guide.

4.4. Saving a Report

When you edit the report definition (see Defining a Report on page 21 for more information), your changes are saved automatically after a few seconds. Other users who have access to the report will see the changes, however the end-users will not see the updated report until it has been published (see Publishing a Report on page 29 for more information).

4.5. Report Revisions

As the report is published and republished (see Publishing a Report on page 29 for more information), Studio maintains a register of the revisions. You can also create a snapshot of your report manually at any time, allowing you to "backup" your report at various stages as you are building it (see Making a Revision on page 26 for more information). You can later view past revisions (see Viewing a Revision on page 26 for more information), compare revisions (past to past and past to current), and also restore (see Restoring a Revision on page 28 for more information). Note that the "restored" version then becomes the current version.

To view the available revisions, on the Report page click the **History of Revisions** icon towards the upper left corner of the page to open the Revisions column.

- For each revision, the date and time the revision was created, and the initials of the user creating the revision, are listed.
- Click the **Revisions options** icon to open a drop-down list of the options available to you for that revision,
- Click the **History of revisions** icon or the **X** button towards the right side of the column to close the column.

4.5.1. Making a Revision

To make a revision:

1. On the Report page, click the **Create Revision** button in the toolbar.

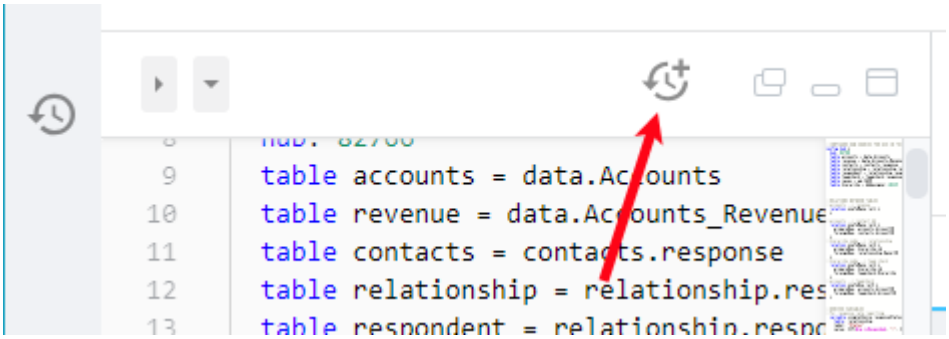


Figure 33 The Create Revision button

A revision is created and added to the list, with the date and time of creation, and the initials of the user who created the revision. This revision will be saved for the life of the report, so you can return to it at any time and view it in read-only mode, copy sections of code from it or restore it as necessary.

4.5.2. Viewing a Revision

When you view a revision it is presented in read-only mode (you cannot edit it) and the preview is displayed. This mode is intended to show how the report looked at the time the revision was created. When you close the Revisions column, the CDL editor will return to the current draft of the code as it was before you opened the Revisions column.

To view a revision:

- Open the Revisions column, scroll through the list of revisions to find the one you need, and click on it to display it in the CDL editor. It will then be loaded in read-only mode.

4.5.3. Comparing Revisions

You may need to compare two revisions of the report code to see for example where changes have been made in the event an error is discovered in the code. To compare revisions:

1. Click the **History of revisions** button to open the Revisions column.

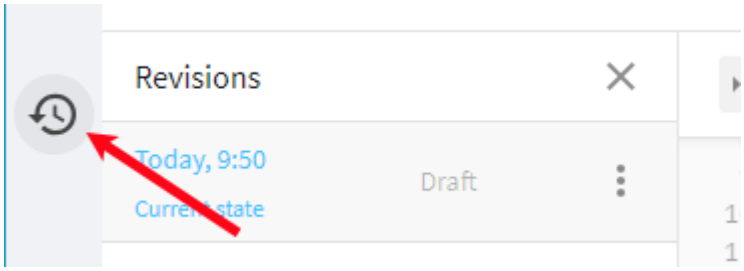


Figure 34 The History of revisions button

2. In the Revisions column, click the **Revisions options** icon ⋮ for one of the revisions you are interested in, and select **Compare** from the drop-down.

   A column of checkboxes opens, with the box beside the selected revision checked.

3. Check the box beside the other revision of interest, then click **Compare** towards the bottom of the column. Note that you can only compare two revisions at one time.

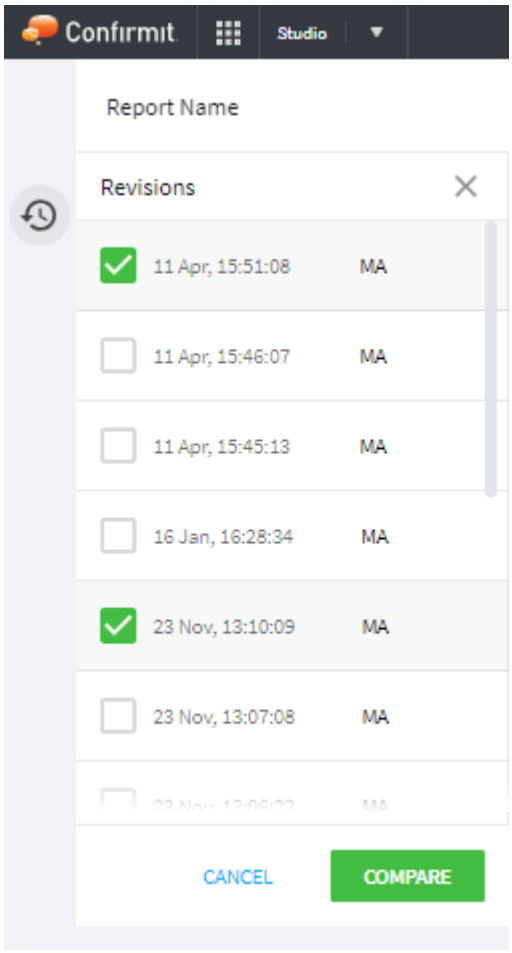


Figure 35 Selecting the revisions to be compared

The Compare overlay opens, with the two revisions displayed side-by-side.

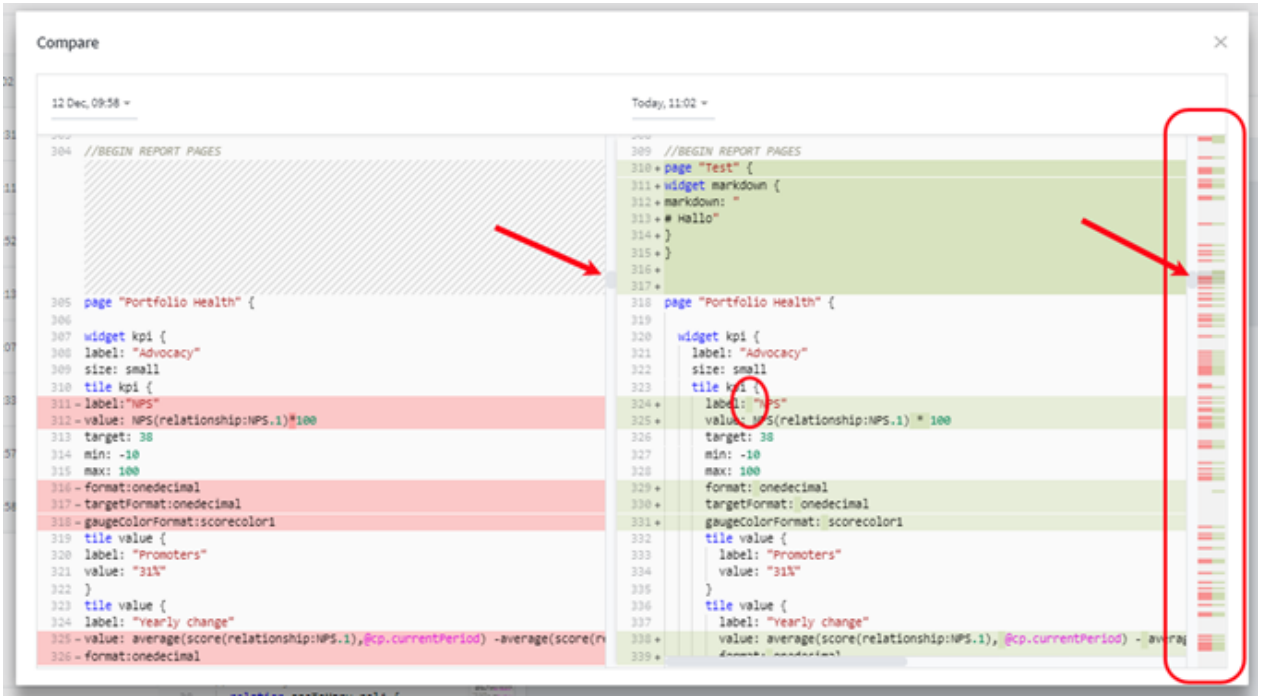


Figure 36 Comparing two revisions

- Any differences between the two revisions are indicated by patches in the columns to the right side of the overlay - ringed above, and the dissimilar rows are shaded.
- Your location within the code is indicated by the shaded 'buttons' in the scroll columns to the right of each revision (arrowed above). Drag these buttons, or if you have a roller mouse use the roller function, to scroll through the code. The two revisions scroll together so you can directly compare any differences.
- The actual changes within the rows are also highlighted in darker shades (an example is ringed in the revision on the right).
- Where one revision has more lines of code than the other (maybe some lines have been added or deleted in one of the revisions), a similar number of shaded lines that are empty and un-numbered are added in the other revision so that code that is in both revisions maintains line parity between the two versions. Note that Lines with a + sign beside the line number are added; lines with a - sign are removed.
- You can change the revisions being compared; click on the date/time field above the revision you wish to change, and select a different revision from the drop-down.
- When you are finished with the comparison, click the **X** button in the upper-right corner of the overlay to close the overlay.

4.5.4. Restoring a Revision

Whenever you publish or take a snapshot of your report, Studio keeps track of the changes and maintains a register of the revisions. You can later restore a past version to being the "current" version if necessary. Note that restoring a revision actually replaces the content of the current draft with the selected revision's content.

To restore a revision:

1. In the Revisions column, click the **Revisions options** icon ⋮ for the revision you wish to restore, and select **Restore** from the drop-down.

   The selected version of the code is restored to being the "current" version. Bear in mind that the previous "current" version that you are replacing will be overwritten; if you may need it then you must publish it or take a snapshot before you restore the selected version.

Note that when you restore a version, this version will not yet be available to the end users; you must first publish this version to make it available.

4.6. Report Status

Until the report has been published, it will have the status Draft version. While it has this status, only users who have access permission to view and maybe edit the report can see it; end users cannot see it. Only once the report is published will it be Live and accessible to end users.

4.7. Publishing a Report

Until a defined report has been published, its status is **Draft**. It is displayed in the Preview Panel of the Report Editor page (see The Report Editor Page on page 9 for more information)

When the Report Designer has published the report, its status changes to **Live**. End-users who have been granted access rights to this report (see Assigning an End User List to a Report on page 5 for more information) will then see it in their Report List.

To publish a report:

1. Enter **Edit** mode. To do this either click the report name or, when in the Report List page, open the **Report Management** menu and select **Edit** (see The Reports List on page 6 for more information).
2. Click **Publish** in the upper-right corner of the Report Editor page.

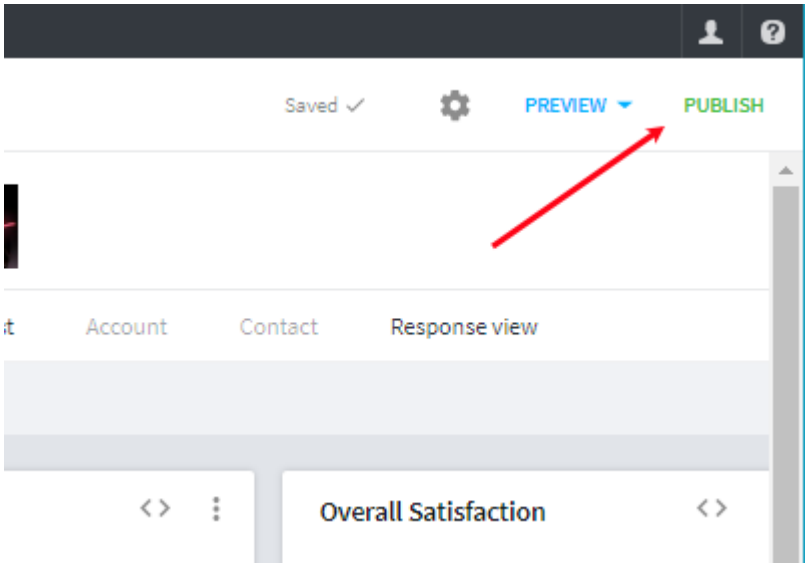


Figure 37 Publishing the report

4.8. Previewing a Report

The Report Designer can preview both the draft and live versions of a report (see Publishing a Report on page 29 for more information) as it will be seen by the End User.

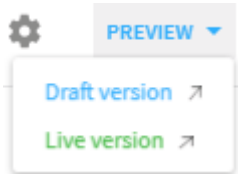


Figure 38 The Preview menu

To preview a report as a Designer:

- When you are in the Edit mode, click the **Preview** icon in the upper-right part of the report page and select the version you wish to see from the menu displayed.

Another method to preview a report that is available to both End Users and Designers is as follows:

- When you are in the Report List page, click the **Report options** icon ⋮ in the relevant row and select **View** to preview the Draft version of the report.

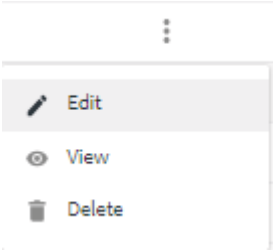


Figure 39 The Report Management menu

To return to the Report List page, click the **Back** button on your browser.

4.9. Printing a Report

You can print out or print to a PDF file the Draft and the Live versions of the report from the Preview page. To do this:

1. In the Studio window, click the **Preview** button and select the version of the report that you wish to print out (see Previewing a Report on page 29 for more information).
2. Sort and filter the report to present the data you wish to print out.
3. Click the **Print** icon in the upper-right corner of the page.

   The Printer details page opens with a preview of the printed output.

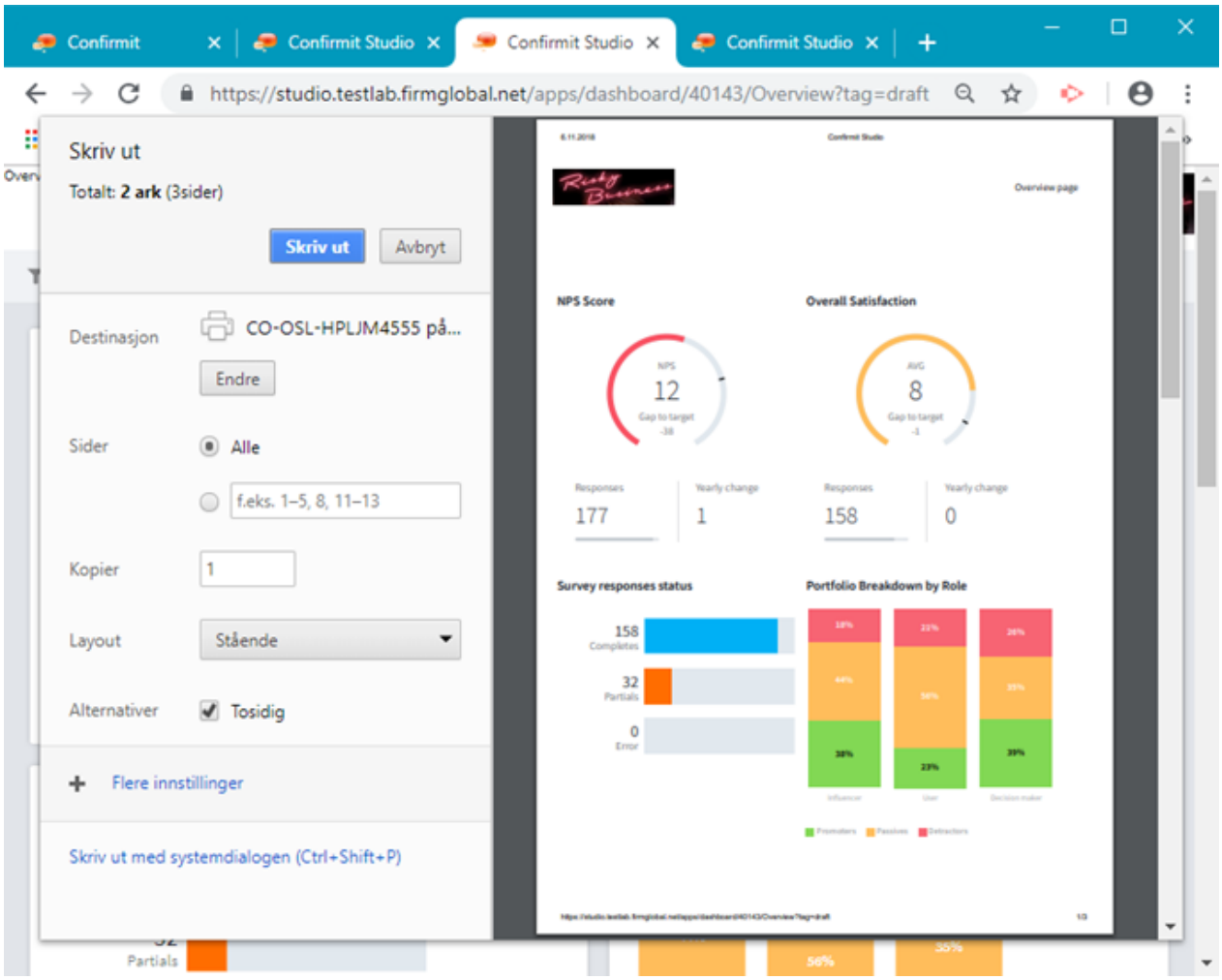


Figure 40 Example of the print preview page for a report

4. Adjust the print settings as required, then click **Print** to complete the operation.

   To save the report as PDF or another format, click **Change** then select the required output format from the list.

Note: The output formats available to you will depend on the browser you are using and what is available via your installation. Print is best supported in Chrome.

4.10. Deleting a Report

To delete a report:

1. In the Report List page, find the report you wish to delete.
2. Click the **Report options** icon ⋮ for that report and select **Delete**.

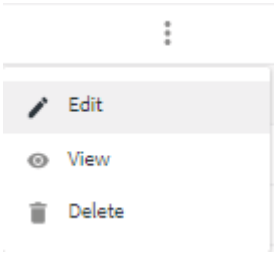


Figure 41 The Report options menu

The delete confirmation overlay is displayed.

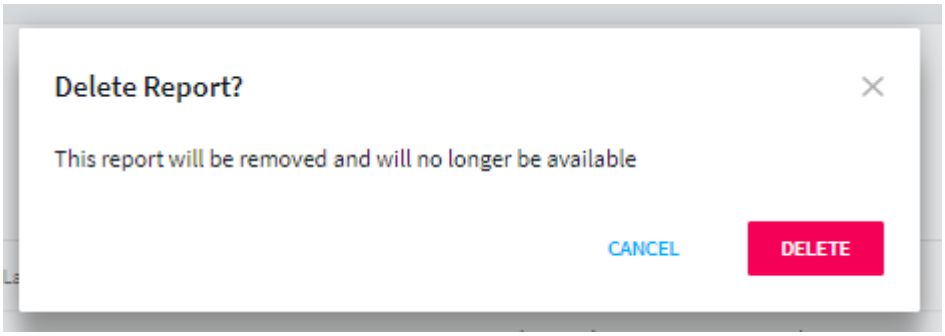


Figure 42 The delete confirmation overlay

3. Ensure you are deleting the correct report, then click **Delete** to delete it or **Cancel** to stop the operation.

5. Variable path (Vpath)

In a Studio report you will often need to reference particular variables, for example to define which results are to be shown in a particular widget. Variables are referenced via a “variable path”, the Vpath. The Vpath syntax is as follows:

```
dataset.table:variable[.fieldId]
```

where

- `dataset` is a Hub dataset, referenced with the "public name" of the dataset. This can be a survey, a custom data table or a contact database.
- `table` is the table or level in the dataset, referenced with its namelevel of the survey. For surveys, these are respondent, response and each of the loops. For contact databases it is response and the surveyactivity loop. For panels it is response and the surveyhistory loop.
- `variable` is the variable that is to be referenced.
- `fieldID` (optional) is the variable's field. This is relevant for variable types that are represented by multiple fields, as for example multi and grid questions in a survey.

Example:

```
p4858585.response:Q1.1
```

This refers to the field with code 1, in a multi or grid question Q1, from the response (survey) data of a survey with survey id p4858585.

6. Expressions in the Data Engine

A Data expression is a function that can be used to retrieve or calculate data from the hub source.

Data can be fetched directly, for example:

```
value: survey:comment
```

or it can be constructed or calculated, for example:

```
// concatenating several string values
value: contact:FirstName + " " + contact:LastName
```

```
// calculating average from LTR question
// aggregations (or any calculations) can only be performed on
categorical questions that have a score
// or numerical questions. So we first need to apply score() function
value: average(score(survey:LTR))
```

```
// calculating average on filtered question
// add up values from AnnualRevenue column in revenue table where
revenue:year value equals 2018
value: sum(revenue:AnnualRevenue, revenue:year=2018)
// number of completed responses
value: COUNT(survey:response,survey:status='complete')
```

6.1. Data Types

The following data types are supported:

Null	Type of null value representing a missing value
Boolean	Represents a boolean value
Integer	Represents an integer number
Real	Represents a floating-point number
Text	Represents a string value
Date/time	Represents a data/time value
Category	Represents a single choice in a categorical variable

6.2. Type Conversions

Values of supported types can be converted from one type to another either implicitly or explicitly. One expression could be variables, constants or functions of different types, and they may need to be converted to something common so that the expression can be evaluated.

- **Implicit** means that one type is converted to another automatically.
- **Explicit** means that you need to write a special instruction in expression to perform the conversion.

Possible conversions are summarized below.

		From						
To		Null	Boolean	Integer	Real	Text	Date/Time	Category
	Boolean	++	++	-	-	-	-	-
	Integer	++	-	++	+	-	-	-
	Real	++	-	++	++	-	-	-
	Text	++	+	+	+	++	+	+
	Date/Time	++	-	-	-	-	++	-
	Category	++	-	-	-	++	-	++

Here:

- indicates no conversion is possible.

+ indicates an explicit conversion is possible.

++ indicates an implicit conversion is possible.

6.3. Expression Elements

Data Engine expressions are built from the following elements:

- Constants. You can define constants of all supported types except category.
- Variable references.
- Operators (unary and binary).
- Function calls.

In some functions the following special elements are required:

- Type reference (for value conversion).
- Level/table references.
- Hierarchy references (for rollup aggregations). A hierarchy is always associated with a particular level.
- Recoding definitions (for Recode function).

6.4. Operators

The expression model supports a number of operators.

6.4.1. Binary Arithmetic Operators

Operator	Operands	Result

Operator	Operands	Result
+	Numeric	Numeric
+ (concatenation)	Text	Text
-	Numeric	Numeric
*	Numeric	Numeric
/	Numeric	Numeric
% (modulo)	Numeric	Numeric

6.4.2. Comparison Operators

Operator	Operands	Result
=	Boolean, Integer, Real, Text, DateTime, Category	Boolean
!=		Boolean
>		Boolean
>=		Boolean
<	Integer, RealDateTime	Boolean
<=		Boolean

6.4.3. Logical Operators

Operator	Operand(s)	Result
AND	Boolean	Boolean
OR	Boolean	Boolean
NOT	Boolean	Boolean
IS NULL	Any	Boolean
IS NOT NULL	Any	Boolean

6.5. Functions

The expression model supports a set of functions. Available functions are listed below.

Most of the functions accept arbitrary expressions (of appropriate type) as arguments. Special cases include:

- Type argument (for type conversion function)
- Recoding definition
- Level reference (for aggregating functions)
- Hierarchy reference (for aggregating functions)
- Ranking criteria (for ranking functions)

The following mnemonics are used in function descriptions:

- a, a1, a2, etc - exressions of any type (or a type specified in comments)
- x, x1, x2, etc - expressions of numeric type (integer or real)
- n, n1, n2, etc. - expressions of integer type
- b, b1, b2, etc. - expressions of boolean type
- s, s1, s2, etc. - expressions of string type
- d, d1, d2 - expressions of date/time type

6.5.1. General Functions

Function	Result type	Comment
Convert(a, type)	type	The value of a converted to the requested type. See type conversion table.
ToInt(a)	Integer	Shortcut for Convert(a,int)
ToReal(a)	Real	Shortcut for Convert(a,real)
ToText(a)	Text	Shortcut for Convert(a,text)
IsNull(a1, a2)	Type of a1	The value of a1 if a1 is not null, otherwise a2. a2 must be convertible to the type of a1
In(a1, a2, ...)	Boolean	Returns true if the value of the first argument is equal to one of the values of a2, etc. The first argument is expected to be an expression of numeric, text or category type. Other arguments are expected to be constants of compatible types.
Between(a1, a2, a3)	Boolean	Equivalent of a1 >= a2 AND a1 <= a3. All arguments must be of (compatible) numeric or date/time type.
Power (x^1, x^2)	Type of x^1	X^1 to the power x^2
Score(c)	Real	Score of the selected category. It is expected that an argument is a (single) variable with scores assigned to its categories.
Abs(x)	The type of x	Absolute value of x

Function	Result type	Comment
Ceil(x)	Integer	Minimal integer greater or equal than x
Floor(x)	Integer	Maximal integer less or equal than x
IIF(b, a1, a2)	Type of a1	If the condition b is true then a1 else a2. b2 must be convertible to to the type of a1
Ln(x)	Real	Natural logarithm of x
Log(x1,x2)	Real	Logarithm of x1 with respect to base x2
Log10(x)	Real	Decimal logarithm of x
Exp(x)	Real	Exponent of x
Sqrt(x)	Real	Square root of x
Round(x, n)	Numeric	Rounded value of x with n decimal places
Recode(a, recoding)	Text	Recodes argument value (of numeric, category, text or date/time type) to a text code according to the recoding definition (see below).
Rank(criteria)	Integer	Evaluates a record rank according to the specified list of sorting criterion
DenseRank(criteria)	Integer	Evaluates a dense record rank according to the specified list of sorting criterion
Demote(a, level)	Type of a	Moves value of first argument to the specified level. Leve must be a successor of the level of the argument.
KeyFilter(level, a1, a2 ...)	Boolean	A shortcut to create a filter expression for level primary keys. The number of key values (a1, ...) must be the same as the number of level key variables.
ChildrenFilter(hierarchy, a1, a2, ...)	Boolean	Evaluates to true if a record is a child record of the specified parent record in the hierarchy. Arguments a1, a2, etc. specify primary key values for the required parent record. If key values are omitted, the function evaluates to true for all root level records.
IsFirst(sourceLevel, targetLevel, sorting)	Boolean	Evaluates to true for every first record from source level related to the same parent record from target level. Each group of records is sorted according to the specified sorting criteria
AnswerText(variable)	Text	Returns the text (title) of current answer of a single variable. Currently a default language (request-level option) is used to retrieve texts.
FieldMask(v, c1, c2, ...)	Type of v (compound)	First argument is expected to be a compound ("vector") value of any type, other arguments must be text constants. The result is a value of the same type that includes only fields with the specified codes.

6.5.2. Date Functions

Function	Result type	Comment
GetDate()	Date/time	Current date and time
Year(d)	Integer	Year part of the specified date/time value
Month(d)	Integer	Month part of the specified date/time value
Quarter(d)	Integer	Quarter number of the specified date/time value
Day(d), (Alias: DayOfMonth)	Integer	Day part of the specified date/time value
DayOfWeek(d)	Integer	Day of week for the specified date/time
Hour(d)	Integer	Hour part of the specified date/time value
Minute(d)	Integer	Minute part of the specified date/time value
Second(d)	Integer	Second part of the specified date/time value
Millisecond(d)	Integer	Millisecond part of the specified date/time value
Week(d)	Integer	Week part of the specified date/time value (see * below)
ShiftedYear(d)	Integer	Year part of the specified date/time value taking into account week rule (see * below)
ShiftedQuarter(d)	Integer	Quarter number of the specified date/time value taking into account week rule (see * below)
CalendarMonth(d)	Integer	Encoded calendar year and month as an integer value (YYYYMM)
CalendarQuarter(d)	Integer	Encoded calendar year and quarteras an integer value (YYYYQQ)
CalendarDate(d)	Integer	Encoded calendar year, month and day as an integer value (YYYYMMDD)
AddYear(d,n) AddQuarter(d,n) AddMonth(d,n) AddDay(d,n) AddHour(d,n)	Date/time	Add the specified number of periods to the date taking into account calendar rules.
DiffYear(startdate, enddate)	Integer	Return the number of period boundaries crossed by the specified date

Function	Result type	Comment
DiffQuarter(startdate, enddate) DiffMonth(startdate, enddate) DiffDay(startdate, enddate) DiffHour(startdate, enddate)		range
InYear(date, from, to, baseDate) InQuarter(date, from, to, baseDate) InMonth(date, from, to, baseDate) InDay(date, from, to, baseDate) InHour(date, from, to, baseDate)	Boolean	Evaluate to true if the specified date is within the specified range of periods. from and to arguments define the number of periods relative to the baseDate. The range always covers whole calendar periods. BaseDate is optional, defaults to current date. In the latter case the functions define rolling time periods. InPeriod(date, from, to, baseDate) is equivalent to Between(DiffPeriod(baseDate, date), from, to)
YearToDate(d) QuarterToDate(d) MonthToDate(d) DayToDate(d) HourToDate(d)	Boolean	Evaluate to true if the specified date is within the specified current period and is less than current date. PeriodToDate(date) is equivalent to date <= GetDate() AND DiffPeriod(date, GetDate()) = 0

*) The Week rule is configured on a query level, and is currently fixed as:

- First day of week is Monday.
- First week starts on the first day of year.

6.5.3. Text Functions

Function	Result type	Comment
Len(s)	Integer	Length of string s
LTrim(s)	Text	Remove leading spaces from s
RTrim(s)	Text	Remove trailing spaces from s
Left(s, n)	Text	Substring of s consisting of n leftmost characters
Right(s,n)	Text	Substring of s consisting of n rightmost characters
Contains(s1, s2)	Boolean	True if s1 contains s2 as a substring
Lower(s)	Text	Lowercased s

Function	Result type	Comment
Upper(s)	Text	Uppercased s

6.5.4. Aggregating Functions

Aggregating functions compute aggregated values from a set of values. All aggregating functions take the following common arguments:

- Value to aggregate.
- Optional target aggregation level (table). Target level must be a (direct or indirect) parent of the source value level. For example in a one-to-many relation between account and contact, account is the parent level. In a one-to-many relation between response and a loop l1, response is the parent level. Note that both levels can be the same, in which case no actual aggregation is performed. The resulting value will be computed per record of this level. See below for "implicit aggregation".
- Optional hierarchy. If a target level is specified it is possible to additionally specify a hierarchy defined on that level. In this case rollup aggregation is performed for each target level record.
- Optional filter expression. It must be an expression of boolean type. If a filter is specified only values satisfying the filter expression are included in aggregation.

There are two forms of aggregation in the Data Engine:

- **Explicit aggregation -** where a target aggregation level is specified. In this case a single aggregated value is evaluated for every record of the target level.
- **Implicit aggregation -** where a target level is not provided. In this case aggregation is performed according to the current context:
    - If the expression is used in a virtual level with groupings specified, then an aggregated value is computed in each group
    - If the expression is used in a child level definition, then an aggregated value is computed for each parent level record
    - Otherwise a single aggregated value is computed (across the whole data set).

A COUNT function is a special case as the value to aggregate can be omitted. In that case a source level must be specified instead. The function will then compute the number of records on the source level.

In the table below common_args refer to all optional arguments: filter, target level/hierarchy.

Function	Argument	Result	Comment
Count(a, common_args) Count(level, common_args)	Any	Integer	Count of non-null values of a Count of records on the source level
Sum(x, common_args)	Numeric	Integer	Sum of x
Average(x, common_args) (alias: Avg)	Numeric	Real	Average value of x
Min(x, common_args)	Numeric or date/time	Type of x	Minimal value of x
Max(x, common_args)	Numeric or	Type of x	Maximal value of x

Function	Argument	Result	Comment
	date/time		
Mode(x, common_args)	Numeric	Type of x	Mode of x
Median(x, common_args)	Numeric	Integer	Median value of x
Variance(x, common_args) (alias: Var)	Numeric	Real	Variance of x
Stddev(x, common_args) (alias: Deviation)	Numeric	Real	Standard deviation of x
StdErr(x, common_args) (alias: Standarderror)	Numeric	Real	Standard error of x
Some(b, common_args)	Boolean	Boolean	True if any of b is true
Every(b, common_args)	Boolean	Boolean	True if all of b is true
RangeGap(x, n1, n2, common_args)	Category	Real	Argument is expected to be a categorical variable with scores assigned. n1 and n2 must be integer constants specifying number of highest and lowest scores to calculate the number of promoters and detractors respectively. Computes the value: (number_of_promoters - number_of_dertactors)/total_response_count Promoters are defined as responses with the n1 highest score values. Detractors are defined as responses with n2 lowest score values.
NPS(x, common_args)	Category	Real	Equivalent to RangeGap(x, 2, 7)
InHierarchy(hierarchy, condition)	Boolean	Boolean	Evaluates to true if any parent record of the current record satisfies the condition
First(x, sorting, commom_args)	Any	Type of x	A pseudo-aggregating function evaluating the first value of the specified expression according to the specified sorting criteria. At the moment only supported only in context of a list query without groupings (cuts). The second argument is expected to be a "sort" node type.

"Weighted" versions of some of the above functions also exist. These versions compute weighted aggregates using weight expression(s) configured at query level. Each weighted function uses a weight specified for the argument's level in the query, if any. If no weight expression is specified, the result is the same as that produced by the respective unweighted function. The list of weighted functions is below:

- WCount
- WSum
- WAvg (or WAverage)
- WStdev (or WDeviation)
- WVar (or WVariance)
- WStderr (or WStandardError)
- WMode

- WMedian
- WRangeGap
- WNps

6.5.5. Vector Aggregating Functions

A number of functions can be used to aggregate values of "vector" (or compound) expressions (such as grids, multi variables, numeric lists, etc.). These functions take a "vector" argument of a specific type, and produce a scalar result from all values in a compound value.

Function	Argument	Result	Comment
any(v)	Boolean vector (multi)	Boolean	True if at least one value in the argument is true
all(v)	Boolean vector (multi)	Boolean	True if all values in the argument are true
vcount(v)	any vector type	Integer	Number of non-null values in the argument
vselected(v)	Boolean vector (multi)	Integer	Number of selected ("true") values in a boolean vector
vsum(v)	Numeric vector type	Source element type	Sum of non-null values in the argument
vaverage(v) (alias vavg(v))	Numeric vector type	Real	Average of non-null values in the argument
vmin(v)	Comparable vector type	Source element type	Minimum of non-null values in the argument
vmax(v)	Comparable vector type	Source element type	Maximum of non-null values in the argument

6.6. Recoding Definition

The Recoding definition object defines mapping rules used for recoding the source value into a set of text codes. A recoding definition consists of one or more recoding rules, each of which maps input values to a specific output code. All rules in a single recoding definition must be of the same type, and they must not overlap.

The following types of recoding rules are supported:

- **text → text** - This type maps a set of input codes to a single code (similar to Reportal variable recoding).
- **integer → text** - This type maps a set of integer values to a single code.
- **range → text** - This type maps a range (numeric or date/time) to a single code. All ranges in a recoding definition must be of the same type. Possible range types are:
    - **open** - both boundaries are not included.
    - **left-open** - left boundary is not included.
    - **right-open** - right boundary is not included.
    - **closed** - both boundaries are included.

6.6.1. Sorting Criteria

Sorting criteria are used in ranking functions. A criterion consists of the following items:

- An Expression defining the value to sort by (must be of comparable type).
- The sorting direction (ascending/descending).

7. Report Definition Structure

The Report definition is organized into blocks of CDL code. These blocks are as follows:

- **Title -** the first line in the report code, contains the report title.
- **Config Hub** - defines and configures the hub from which the data that is to be presented in the report is sourced.
- **Config Report** - defines and configures fundamental settings for the report. This includes blocks for:
    - **Config Access** - sets up access permissions for the various users and end users.
    - **Filter pane** - defines the filters that are to be available for the users and end users.
- One or more **Page elements** - your report should be divided into pages to improve presentation and readability. Widgets must be placed on pages.
- One or more **Widget elements** - the data extracted from the hub is presented in widgets. Each type of widget presents a particular type of data in a particular way.

Examples of these blocks are given below:

1. **Title** - the title block is simple and generally contains only one line of code:
```
title "Studio Report 1"
```
2. **Config Hub -** contains the source data configuration. The following parameters are defined in this block:
    - **Hub -** the ID of the hub used for the report.
    - **Table -** table definitions (see Table Aliases on page 48 for more information).
    - Relation oneToMany <name> and Relation oneToOne <name> - this defines the relations between tables, and what fields are used to join data (see Table Aliases on page 48 for more information).
    - Derived variables (see Derived Variables on page 49 for more information).

    For example:
```
config hub {
hub: 555
table accounts = crmdata.externalAccounts
table survey = p555555.response
table contacts = p6666666.response
table healthCheck = p7777777.response
table cases = am.case
table respondent = p5555555.respondent
table revenue = crmdata.Historical_Revenue

relation oneToMany rel2 {
primaryKey: accounts:AccountID
foreignKey: contacts:AccountID
}
}
```
3. **Config Report -** contains the report configuration. The following parameters are defined in this block:
    - **Logo -** gives the path to the logo image file.
    - **Formatter -** defines properties of formatters (see Formatters on page 53 for more information).
    - **View -** defines properties of views (see Views on page 57 for more information).

```
config report cr {
logo: "/isa/BDJPFRDMEYBPBKLVADAYFQCDAVIOEQJR/logo_28x140.png"
```

```
formatter number formatterLTR {
numberDecimals : 1
decimalSeparator : "."
}
formatter number formatterRR {
numberDecimals : 0
decimalSeparator : "."
shortForm : true
postfix : "%"
}
formatter number customEmpty{
numberDecimals : 0
emptyValue: -
}
formatter objectProperty textPicker {
property: text
}
formatter color backgroundColor {
thresholds: #e8f8e0 >= 100%, #ffeed6 >= 80%, #fedfe2 >= 0%
}
formatter color valueColor {
thresholds: #388e3c >= 100%, #ff6d00 >= 80%, #d40000 >= 0%
}
formatter date DDMMMYYYY {
format: "DD MMM YYYY"
shortForm: true
emptyValue: -
}
formatter date dateRelative {
locale:en
shortForm: false
relative:true
}
formatter text commentFormat {
useDots:true
length:68
emptyValue: -
}
view metricWithChange metrics {
backgroundColorFormatter: backgroundColor
valueColorFormatter: valueColor
fontSize:medium
roundCorners:true
}
view camelCSS subheader {
color : "rgba(18, 24, 33, 0.54)"
fontSize: "14px"
}
view icon icon {
size: "60"
roundedCorner: true
}
view link openLink {
label: "Open link"
}
```

Access – contains the configuration of end user access to the report.

```
config access {
portalId: 11111
}
```

Filter - contains the filter configuration.

```
filter multiselect {
```

```
label: "Account Rating"
option checkbox {
label:"Gold"
value:accounts:TotalAccountValue > 200000
}
option checkbox {
label:"Silver"
value: accounts:TotalAccountValue >99999 AND accounts:TotalAccountValue
<199999
}
option checkbox {
label:"Bronze"
value: accounts:TotalAccountValue < 100000
}
```

4. **Page** - specifies the page configuration. This consists of the page type and title, and one or more widget configurations.

```
page account "Account" {
widget search {
table: contacts:
layoutArea: "header"
value: @cr.fullContactName
navigateTo: "Contact"
}
```

5. **Widget -** this block contains the widget definition. Any number of widgets can be included in a page, though the number should be limited to ensure clear presentation of the data. A widget block must be located within a Page block.

```
widget search {
table: contacts:
layoutArea: "header"
value: @cr.fullContactName
navigateTo: "Contact"
}
```

7.1. Title

The Title is the first line in the report code, and defines the name of the report.

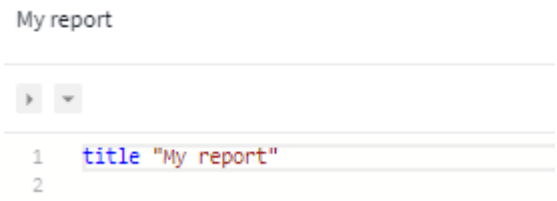


Figure 43 Example of the Title code line

The title line is created automatically, complete with the report name, when you first create the report; you must name the report before you can create it. Note that you can change the name of the report later (see The Report Editor Page on page 9 for more information). Note also that you do not change the report title by editing the 'title' code in the CDL Editor panel.

7.2. Config Hub

The **config hub** code block - specifies the data that is to be used in the report, defines the relations between different data tables, and defines any derived variables (see Configuring the Hub on page 22 for more information).

```
config hub {
    hub: 127220
    table revenue = crmdata.HRev  // historical revenue custom table.
    table accounts = crmdata.Accounts
    table survey = p6578656.responseid
    table respondent = p6578656.respondent
    table contacts = p7457531.responseid
    table cases = am.case
    table healthCheck = p6555465.responseid

  //accounts --> revenue

  relation oneToMany rel3 {
    primaryKey: accounts:accountID
    foreignKey: revenue:accountID
  }

  // accounts --> contact db --> survey
  relation oneToMany rel4 {
    primaryKey: accounts:AccountID
    foreignKey: contacts:AccountID
  }
```

Figure 44 Example of part of the config hub code for a report

Each Studio report can reference one hub in SmartHub, and this hub must be stated in the code, referencing the hub id (arrowed above).

Important
You are strongly recommended to add comments into the CDL at every opportunity (see Comments in CDL on page 16 for more information).

7.2.1. Table Aliases

All tables in the referenced hub are available for use, and can be referenced in the report using vpath (see Variable path (Vpath) on page 33 for more information). To simplify referring to particular variables inside a hub, you can define a table alias to refer to a particular table instead of using the full path. This also simplifies reuse of the CDL from one report on other datasets in another report, as you will then only have to change the reference in the hub configuration section, not everywhere a table is referenced.

The general syntax for defining a table alias is:

```
table table_alias_name = datasetname.tablename
```

```
config hub {
    hub: 127220
    table revenue = crmdata.HRev  // historical revenue custom table.
    table accounts = crmdata.Accounts
    table survey = p6578656.response
    table respondent = p6578656.respondent
    table contacts = p7457531.response
    table healthCheck = p6555465.response
    table cases = am.case
```

Figure 45 Example of a table alias

For example, the table specified in row 5 above - **cdmdata.HRev** - which is a custom table HRev set up in SmartHub in the data set cdmdata, is here given the alias **revenue**. So from now on in this report, whenever you want to use the data from the **cdmdata.HRev**, you only need to use the name **revenue**.

For surveys, combined surveys, panels and contact databases, the dataset name is by default the survey id, for example:

```
table survey = p123456.response
table respondent = p123456.response
table contacts = p654321.response
table healthCheck = p12323434.response
```

Note: SmartHub includes functionality for changing the name of the data set from the default (project id) to something else. This is called the "Public name", and is done under the Settings menu in SmartHub. This capability allows you to set names that are easier to relate to than a project id, and can be also be used if you need to switch out entire data sets in a simple manner.

For tables with hierarchical data, the following syntax is used:

```
table hier = dbdesigner.HierarchyID
```

where HierachyID is the id of the hierarchy from Hierarchy Management (refer to the separate Hierarchy Management User Guide for detailed information), or from the hierarchy tab in the Schema in Database Designer (refer to the separate Confirmit Professional Authoring User Guide for detailed information).

7.2.2. Relations

When a one-to-many relation is to be defined (for example, one account to many contacts), for the data engine to know how to join the relevant tables, the following syntax is used:

```
relation oneToMany rel1 {
primaryKey: accounts:AccountID
foreignKey: contacts:AccountID
}
```

where `primaryKey: accounts:AccountID` is the path to the primery key, here the account ID field in the account table, and `secondaryKey: contacts:AccountID` is the path to the foreign key, here account ID field in the contact table.

The contacts are linked to an account through the foreign key, with potentially multiple contacts being linked to the same account (see Table Relations in a Studio Report on page 23 for more information).

A one-to-one relation uses a similar syntax:

```
relation oneToOne rel1 {
primaryKey: accounts:AccountID
foreignKey: response:AccountID
}
```

Note: Using a one-to-one relation will fail if there is actually more than one record with a particular key. So use oneToMany unless you are 100% sure that the one-to-one relation is correct.

7.2.3. Derived Variables

A Derived variable is a virtual variable that you can create and add to the hub source by calculating or categorizing already existing variables. It is defined in CDL, and can be used elsewhere in a report as if it were a "normal" variable existing in the hub. A derived variable can be referenced in an expression of another derived variable. This concept can help you to avoid copy/pasting of the same formulas if you need to use the same data expression on more than one occasion, and you can also avoid having to conduct data processing (or other pre-processing procedures) before you can use such values in your report.

Derived variables are defined in the hub configuration block of the report definition (see Defining a Report on page 21 for more information)

The following types of derived variables are available:

- **Single-choice**:

```
//single-choice derived variable based on date variable in survey
variable singleChoice b2bPeriodsDerived {
table: survey:
label: "b2b Periods derived"
option code {
label: "Before 22nd of June 2016"
code: "1"
}
option code {
label: "After 22nd of June 2016"
code: "2"
}
value: IIF(survey:interview_start > 2016-06-22,"2","1")
}
```

```
//single-choice derived variable based on numeric variable in custom data
variable singleChoice accountRatingDerived {
table: accounts:
label: "Account Rating Derived"
option code {
label: "Gold"
code: "1"
}
option code {
label: "Silver"
code: "2"
}
option code {
label: "Bronze"
code: "3"
}
value: IIF(accounts:TotalAccountValue > 200000, "1",
IIF(accounts:TotalAccountValue < 100000, "3", "2"))
}
```

```
//single-choice derived variable based on single question in survey
 variable singleChoice riskValueDerived {
table: survey:
label: "Risk Value derived"
option code {
label: "Low derived"
code: "L"
score: 1
}
option code {
label: "Medium derived"
code: "M"
score: 2
}
option code {
label: "High derived"
code: "H"
score: 3
}
value:
IIF(In(survey:Q1,"0","1","2","3","4","5"),"H",IIF(In(survey:Q1,"9","10"),
"L","M"))
}
```

```
//single-choice derived variable based on a derived variable (see below)
variable singleChoice ltrVsRating {
label: "ltrVsRating"
table: accounts:
option code {
label: "Urgent"
code: "1"
}
option code {
label: "Attention"
code: "2"
}
value: IIF((average(survey:ltrDerived) < 7 AND
accounts:accountRatingDerived = "1"), "1", "2")
}
```

```
//single-choice derived variable base on vector
variable singleChoice productFeedback {
label: "productFeedback"
table: survey:
option code {
label: "High"
code: "h"
}
option code {
label: "Medium"
code: "m"
}
option code {
label: "Low"
code: "l"
}
value: IIF(ANY(score(survey:q9)>6), "h", IIF(ANY(score(survey:q9)<4),
"l", "m"))
}
}
```

- **Auto** (includes all types of variables such as integer, real, text, date/time, boolean):

```
//auto derived variable based on score from single question in survey
variable auto ltrDerived {
table: survey:
value: score(survey:Q1)
label: "LTR derived var"
}
//auto derived variable based on variable in custom data
variable auto renewalMonthDerived {
table: accounts:
value: accounts:RenewalMonth
label: "Renewal Month derived var"
}
```

The multi choice question type is currently not supported.

7.2.4. User Property Claim And Access Rules

You can grant access rights to pages and/or widgets based on the properties of a user (claims). These can be either claims the user gets from an identity service, or custom claims retrieved from data in the hub, for example in a tables as below in custom data.

Username	Role	UnderscoredName	SalesRegion	Region

Username	Role	UnderscoredName	SalesRegion	Region
borte;	Manager;	Borte_Sken;	4;	Nordic_VoC;
filis;	AccountOwner;	Filis_Bung;	4;	Nordic_VoC
magdas;	AccountOwner;	Jeck_Burger;	9;	US_VoE;

Figure 46 Example of a claim table

A report can have several user claims. A user claim is defined by using `user property` of type `claim`, and is referenced by its name by the `access rule` when applied at the page or widget level. The user claim properties are as follows:

- **name -** used by the `access rule` to reference the claim when the rule is applied to a page/widget.
- **joinKey** - the vpath(see Variable path (Vpath) on page 33 for more information) to the username column of the table. The username links the logged-in user to the corresponding user claim(s) stored in the same table or different tables.

Important
The value of the username column must match the username of the currently logged-in user.

- **value** - the vpath(see Variable path (Vpath) on page 33 for more information)to the claim column of the table.

```
userProperty claim myRole { // adds a new value on @currentUser object
that we can refer to later e.g. in filter
joinKey: accessRules:Username // username to match on currently logged in
user
value: accessRules:Role // which corresponding column value we want to
use
}
userProperty claim myFullName {
joinKey: accessRules:Username
value: accessRules:UnderscoredName
}
userProperty claim myRegion{
joinKey: accessRules:Username
value: accessRules:SalesRegion
}
userProperty claim myRegionName{
joinKey: accessRules:Username
value: accessRules:Region
}
}
```

The `access rules` applied to a report or a page represent a set of `rule claims`, each of which corresponds to one of the claims. The rule claim properties are as follows:

- **name** - the name of the claim used by the rule claim;
- **value** - the expected claim value the logged-in user must match to see the page/widget.

Note: Rule claims are combined using the AND operator, so every rule claim must be evaluated to TRUE for the user to allow the end user to see the page\widget.

```
access rules { // this widget will be visible to user with
myRole="Manager" AND myRegionName="US_VoE"
rule claim {
name: "myRole"
value: "AccountOwner"
}
rule claim {
name: "myRegionName"
```

```
value: "US_VoE"
}
```

7.3. Config Report

The Config report code block contains the code that configures the report itself. Here you set up any views you may need, maybe set your company's logo on the report, and define for example number and date formats, thresholds and colors to be used. Here you also include the code to enable exporting the report to PDF if this functionality is required (see Exporting to PDF on page 106 for more information).

```
115  config report cr{
116    logo: "/isa/BDJPFRDMEYBPBKLVADAYFQCDAVIOEQJR/magdalenas/riskylogo.jpg"
117
118    formatter date date11 {
119      formatString: "DD MMM YYYY"
120      emptyValue: 'None'
121    }
122    formatter color valueColor {
123      thresholds: #388e3c >= 8, #ff6d00 >= 6, #d40000 >= 0
124    }
125    formatter value valueFMT {
126      emptyValue: " "
127    }
128      formatter number formatterID {
129        numberDecimals : 0
130        prefix : "$ "
```

Figure 47 Example of part of the Config report code block

The following properties are defined in the report configuration block:

- **Views -** one or more views can be configured in a report (see Views on page 57 for more information).
- **Formatters -** one or several formatters can be configured in a report (see Formatters on page 53 for more information).
- **Logo -** this would be a relative link to an image file of the company's logo. This image file must be located in the File Library, and will be displayed in the top part of the report (see Logo on page 59 for more information).

Important
You are strongly recommended to add comments into the CDL at every opportunity (see Comments in CDL on page 16 for more information).

7.3.1. Formatters

Each type of property or value displayed in a widget has a corresponding formatter that defines how that value or property is to look or behave. These can include color, font size, the character to be used as a separator, the action to be taken in the event the property is not given a value etc.

You create and define a formatter in the report configuration block, and then refer to it later in the report when you want to specify particular properties, looks or actions for specific parts of a widget. You will probably need several versions of a particular type of formatter, for example to set different layouts on numbers in the various columns in a widget, or to set different colors onto different parts of a widget, so each formatter of a particular type must have a unique name.

```
}
  formatter number formatterID {
    numberDecimals : 0
    prefix : "$ "
    decimalSeparator: "."
    integerSeparator : " "
    shortForm : true
}
  formatter number formatterLTR {
    numberDecimals : 0
     decimalSeparator : "."
     shortForm : true
     emptyValue: "-"
}
  formatter number formatterRR {
    numberDecimals : 0
    decimalSeparator : "."
    shortForm : true
    postfix : " %"
}
```

Figure 48 Example of several formatters of the same type, with different names

7.3.1.1. Number Formatter

Use the number formatter to specify how numeric values are to be presented.

```
formatter number UScurrency {
  numberDecimals: 1
  prefix: "$ "
  decimalSeparator: "."
  integerSeparator: ","
  shortForm: true
}

formatter number nodecimal {
  numberDecimals: 0
  decimalSeparator: "."
  shortForm: true
  emptyValue: "-"
}

formatter number onedecimal {
  numberDecimals: 1
  decimalSeparator: "."
  integerSeparator: ","
  shortForm: false
}

formatter number twodecimal {
  numberDecimals: 2
  decimalSeparator: "."
  integerSeparator: ","
  shortForm: false
}
```

Figure 49 Example of the number formatter code

The number formatter will not fail on text strings. If the value is an empty string, the number formatter will fall back to the value specified in `emptyValue` (if one is present) or a formatted zero integer. If the value is not an empty string, the formatter will display the string as-is. This behavior is possible when expressions with `IIF` are used in table query and return strings.

Properties	Description	Default value
shortForm	When set to true, K / M / G is used for large numbers.	false
prefix	Text to insert before the numeric value	
postfix	Text to insert after the numeric value	
numberDecimals	Defines the number of digits displayed after the decimal point	2
integerSeparator	Defines the character used as a thousands separator (in this example, a space)	" "
emptyValue	Defines the string to be displayed for an empty value (0, "", null or undefined)	
decimalSeparator	Defines the character used as a decimal separator	

Note: To display an empty cell, use the string: emptyValue: ' '

7.3.1.2. Text Formatter

The text formatter allows you to define how texts are to be displayed. You can for example set it to truncate the text after it reaches the number of characters preset by `length`. Then if `useDots` is set to true, the ellipses character "..." is placed after the truncation point. If the text is an empty string and you want to display something there so the field isn't totally blank, type the required text or characters into the `emptyValue` property.

```
formatter text formatterTT {
  emptyValue: "-"
  length: 25
  useDots: true
}
```

Figure 50 Example of a text formatter

In the example code above, if a text string is longer than 25 characters it will be truncated and the excess will be replaced by the **...** character, and if the field is blank then a **-** character will be displayed.

Properties	Description	Default value
useDots	Defines whether the text is trimmed to a specified length	true
length	Defines the length to trim to. Allowed range is from 3 to infinity.	
emptyValue	Defines the string displayed for an empty value (0, "", null or undefined)	

7.3.1.3. Date Formatter

The Date formatter formats an ISO_8601 date to a human-readable form. If the date string is invalid (it is not in ISO_8601 format or is empty) and the `emptyValue` property is undefined (see below), it will print `Invalid date`. If you wish to customize the output when the value is empty, specify a value for the `emptyValue` property.

```
formatter date date11 {
  formatString: "DD MMM YYYY"
  emptyValue: 'None'
}
```

Figure 51 Example of a date formatter

Properties	Description	Default value
locale	Defines the locale to be used	'en'
shortForm	Defines whether the date will be displayed in a short form for an absolute date value. For example, **17 Jan '17** instead of **17 January 2017**, or **15hrs ago** instead of **15 hours ago** when using a relative date value (below).	true
relative	Defines whether a relative date is to be used.	false
formatString	Allows you to define a custom date value format.	
emptyValue	Defines the string to be displayed for an empty value (0, "", null or undefined).	

Note: The formatString property overrides the shortForm property.

7.3.1.4. Color Formatter

This formatter defines the threshold values for when the colors used in the various dials and graphs in the widgets are to change, and the associated colors that are to be used for each range of values.

```
formatter color scorecolor1 {
  thresholds: #7ED251 >=4 , #F0AD4E >=2, #FA5263 >=1
}

formatter color percentcolor {
  thresholds: #82D854 >= 100%, #FFBD5B >= 70%, #FA5263 < 70%
}

formatter color calloutred {
  thresholds: #ff0000 >=1, #31363e >=0
}

formatter color NPScellcolor {
  thresholds: #5CB85C >= 40, #F0AD4E >= 15, #FA5263 >= 0
}

formatter color NPStextcolor {
  thresholds: #333333 >= 40, #333333 >= 15, #ffffff >= 0
}
```

Figure 52 Example of the formatter color code

To define the thresholds, you start with the highest threshold and specify the color that will be used when values are greater than or equal to that threshold. You then move to the 2nd highest threshold and specify the color that will be used when values are greater than or equal to that threshold, and so on. List all the desired thresholds and colors, with each threshold separated by comma.

In the first example ringed above, in a widget where **scorecolor1** is to be used, for data values greater than 4 the color coded #7ED251 will be used, for values less than 4 but greater than 2 the color coded #F0AD4E will be used, and for values between 2 and 1 the color coded #FA5263 will be used.

No default values are defined.

7.3.1.5. Value Formatter

This is a default value formatter that performs no transformation to the value. When the value is null or undefined ("", null, undefined, or 0 in the case of the number formatter), it will return the string specified in the `emptyValue` property.

```
formatter value emptyvalue {
  emptyValue: " "
}
```

Figure 53 Example of the Value formatter

Properties	Description	Default value
emptyValue	Defines the string displayed for an empty value (0, "", null or undefined)	

7.3.2. Views

A view is a set of properties that define the style of the cell content. Views may contain formatters (see Formatters on page 53 for more information)

```
263     }
264     view metric metrics {
265       valueColorFormatter: valueColor
266       fontSize:large
267       backgroundColorFormatter: transparent
268     }
```

Figure 54 Example of a View code block

7.3.2.1. Metric

This is a type of view used to display a metric.

Property	Purpose	Data Type	Example/Default Value
backgroundColorFormatter:	Defines the background color	id or string	
valueColorFormatter:	Defines the value color	id or string	
chartColorFormatter:	Defines the chart area color	id or string	
fontSize:	Defines the font size	string	small / middle / large

7.3.2.2. MetricWithChanges

This is a type of view used to display a metric together with an arrow or flat indicator denoting the corresponding value change or a lack thereof.

Property	Purpose	Data Type	Example/Default Value
backgroundColorFormatter:	Defines the background color	id or string	
valueColorFormatter:	Defines the value color	id or string	
chartColorFormatter:	Defines the chart area color	id or string	
fontSize:	Defines the font size	string	small / middle / large

7.3.2.3. CamelCSS

This is a preset type of view that allows you to style an item or text in a tile with cascading style sheets (CSS) using "camelCase". Camel case is the practice of writing compound words such that each word or abbreviation in the middle of the phrase begins with a capital letter, for example camelCase.

7.3.2.4. Comments

This is a type of view for the display of open text responses. The number of response text lines to be displayed is by default 3 - edit this here. The default view settings may be changed and extended.

Property	Purpose	Data Type	Example/Default Value
lines:	Defines the number of lines for multi-line text	number	3

7.3.2.5. ExpirationProgess

This is a type of view that defines a progress bar that indicates how many days remain before the expiration date. This view is used to override the default settings.

Property	Purpose	Data Type	Example/Default Value

7.3.2.6. Icon

This is a type of view used to display an icon.

Property	Purpose	Data Type	Example/Default Value
size:	Defines the size.	string	small / middle / large
roundCorners:	Defines whether the widget has round or square corners.	boolean	

7.3.2.7. IconText

This is a type of view used to display an icon with text.

Property	Purpose	Data Type	Example/Default Value
size:	Defines the size.	string	small / middle / large
roundCorners:	Defines whether the widget has round or square corners.	boolean	

7.3.2.8. Link

This is a preset view configuration used to display a navigation link.

Property	Purpose	Data Type	Example/Default Value
label	Defines the label	string	small / middle / large
newTab:	Defines whether the item navigated to will be opened as a new tab.	boolean	

7.3.3. Logo

You can add a company's logo to the report by specifying the relative link URL to the logo image in the `config report` block. The image file must previously have been added to the File Library - refer to the separate Professional Authoring User Guide for further details.

```
115
116  config report cr{
117    logo: "/isa/BDJPFRDMEYBPBKLVADAYFQCDAVIOEQJR/magdalenas/riskylogo.jpg"
118
```

Figure 55 Example of a URL to a logo image file located in the File Library

7.4. Pages

Studio report pages are displayed as tabs both in the Design Area of the Report Editor for Designers and on the report page for end users.

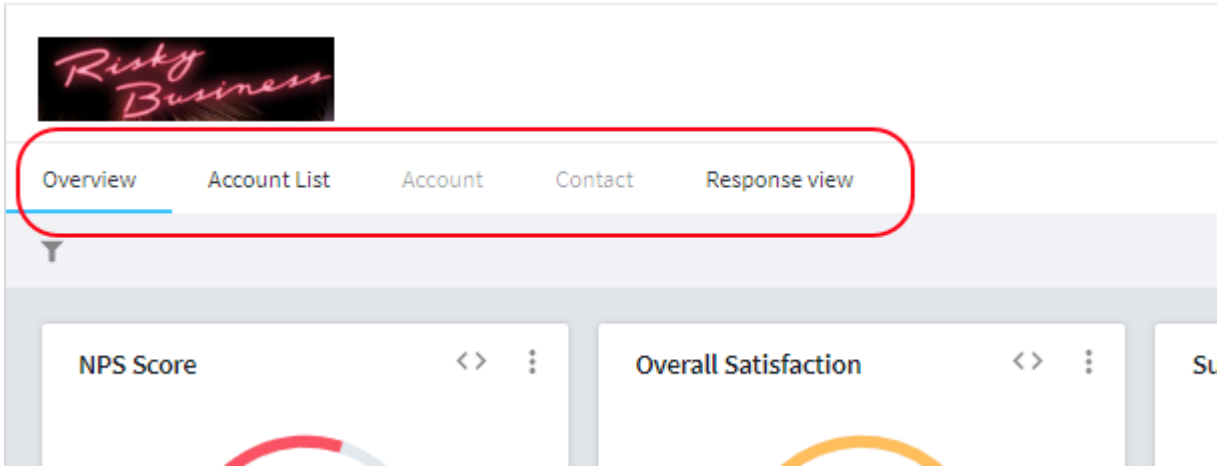


Figure 56 Example of page tabs in a report

Studio has two basic page types:

- The default "generic" page that is rendered when no page type has been specified for the page in the page configuration.
- Pages of specific types that must be explicitly specified. These pages usually have custom properties set that differ from the generic page type. These specific types are:
    - **Account** - pages of this type are designed to display account-related widgets as part of an account health solution.
    - **Contact** - pages of this type are designed to display contact-related widgets as part of an account health solution.

Widgets are contained on pages, and pages can be navigated to from widgets. In the figure below, the kpi widget with the name *NPS Score* is placed on the *Overview* page.

Figure 57 Example of a widget on a page

Pages can be hidden based on the user access rights, or until a value is selected on another page. For example, if the end user must select an account in an Account List page before they can navigate to an Account Details page, then the Account Details page can be hidden until an account is selected.

7.4.1. Config Layout

Note: horizontalAlignmentMode is currently the only valid property for the config layout section.

The horizontalAlignmentMode property is used to support a fixed-width, narrow or centered layout, as used by for example the Pulse solution. This property centers the widgets and the content of the header and toolbar in the dashboard area. The property is added in the configuration section: config layout. The property has the following options:

- **fourColumnsCentered**
- **threeColumnsCentered**
- **twoColumnsCentered**
- **fullWidth** (default) where the layout stretches to fill the width (100%) of the view-port

An example of the property in use in CDL:

```
config layout {
horizontalAlignmentMode: twoColumnsCentered
}
```

And this produces a layout as:

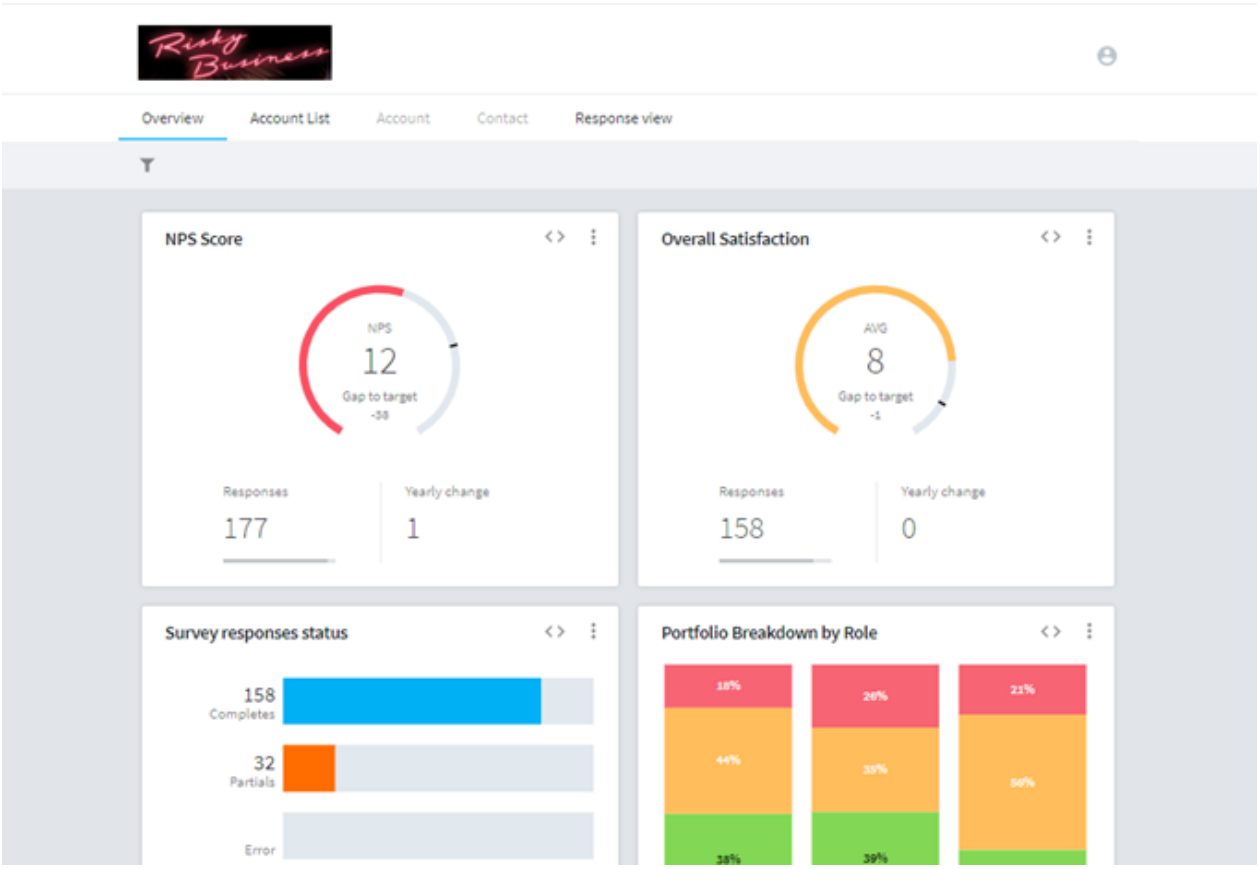


Figure 58 Example of the config layout property in use

The table below shows the number of widgets that will be presented across a page, in the three widget sizes available, for the three horizontalAlignmentMode options.

options (nr. of columns)	nr. of widgets of size small	nr. of widgets of size medium	nr. of widgets of size large
twoColumnsCentered	2	1	1
threeColumnsCentered	3	1 medium + 1 small	1
fourColumnsCentered	4	2	1

Number of widgets across a page

7.5. Filters

You can configure a number of different types of filter for your Studio report. The following filter types are currently available:

- **Multiple choice filter (called multiselect) -** this type includes checkboxes, allowing the user to select more than one option.
- **Single choice filter (called singleselect) -** this type has radio-buttons allowing the user to select only one option at a time. This type requires one option to be selected by default, so it will usually include a catch-all option that doesn't perform any filtering.
- **Hierarchy filter -** this is a variation of the multiselect filter. It is intended to be used with hierarchies from DB designer and drill-down paths defined in Studio, and functions such that when a hierarchy node is selected, all the sub-nodes are also selected.

These filters will then be available in the report so the end user can select the data that is to be included (see Using Filters in the Report on page 69 for more information).

Note: When a user applies a filter to a report, it will be active only for the current session. If the session is closed for any reason (the user closes the report, the session times out etc.) then the filter will be removed from the report.

7.5.1. Basic Filter Setup

To make a filter visible in the report, the filter code must be placed inside a special block called layoutArea toolbar {}, as in the example below:

```
55  layoutArea toolbar {
56    filter multiselect accountOwenr {
57      label: "Account Owner"
58      optionsFrom: accounts:AccountOwner
59    }
60
61    filter multiselect region {
62      label: "Region"
63      optionsFrom: accounts:SalesRegion
64    }
65    filter multiselect name {
66      label: "Name"
67      optionsFrom: accounts:SalesRegion
68    }
```

Figure 59 Example of the layoutArea toolbar code

The funnel icon will then appear in the report toolbar, allowing the user to access the filter panel.

7.5.2. Filter Examples

The following sections describe a number of filter options, giving examples of the code in each case.

7.5.2.1. Fetch All Options

The simplest type of filter can be set up to merely fetch all the available options from a categorical question.

Note: This type works only for categorical questions (for example a single question).

```
layoutArea toolbar {
filter multiselect {
optionsFrom: survey:NPSSegment
}
}
```

7.5.2.2. Custom Filters Based on Data Expressions

If you want to create some special filter categories based on values from question, you can build your own filter.

Basic custom filter properties

```
filter filterType [identifier] {
// where filterType ::= singleselect | multiselect | hierarchy
// identifier is optional, if you want to refer to the filter value it
needs to
have unique identifier
label: "Filter label"
option optionType [optionIdentifier] { // where optionType ::= checkbox |
radio
, depending on filterType
label: "Option 1" // optionIdentifier is optional
value: condition // e.g. survey:q1 > 7
}
...
}
```

Singleselect example

```
layoutArea toolbar {
filter singleselect period {
label: "Reporting period"
option radio s1 {
label: "Last quarter"
value: InQuarter(survey:interview_start, -1, 0)
}
option radio s2 {
label: "Last 6 months"
value: InMonth(survey:interview_start, -6, 0)
}
option radio s3 {
label: "Last 11 months"
value: InMonth(survey:interview_start, -11, 0)
}
option radio s4 {
label: "Last 12 months"
value: InYear(survey:interview_start, -1, 0)
}
}
}
```

Multiselect example

```
layoutArea toolbar {
```

```
filter multiselect {
label: "Account Rating"
option checkbox {
label: "Gold"
value: accounts:TotalAccountValue > 200000
}
option checkbox {
label: "Silver"
value: accounts:TotalAccountValue >99999 AND accounts:TotalAccountValue
<199999
}
option checkbox {
label: "Bronze"
value: accounts:TotalAccountValue < 100000
}
}
}
```

Hierarchy filter example

```
config hub {
hub: 1169
table hier = dbdesigner.364 // table id
table survey = p1079889.response
// relation between hierarchy and survey needs to be defined like between
any
other two tables
relation oneToMany rel4 {
primaryKey: hier:id
foreignKey: survey:q1
}
}
layoutArea toolbar {
filter hierarchy {
label: "Hierarchy filter label"
hierarchy: hier:174 // hierarchy id
optionLabel: hier:language_text
root: 45
default: 45
}
}
```

You can set a default selection for a hierarchy filter. To do this, use the **default** property and refer to the id of the hierarchy node you want to set.

You can also only make a part of the hierarchy available for filtering, for example a particular country, by using the root property. Then the rest of the hierarchy will be hidden, and the user will only be able to set a filter from that node and down in the hierarchy.

These properties can also be fetched dynamically from a hub table, for example Node Assignments from Hierarchy Management, to provide personalized filter capabilities, in this example when reporting off employee data in a contact database:

```
config hub {
hub: 129227
table hierarchy = dbdesigner.19423
table userNodes = Hierarchy11148.NodeAssignments
table employees = p1867705701.response
relation oneToMany {
primaryKey: hierarchy:id
foreignKey: employees:hierarchy
}
userProperty claim nodeId {
joinKey: userNodes:UserName
value: userNodes:NodeId
```

```
}
}
layoutArea toolbar {
filter hierarchy {
label: "Hierarchy"
hierarchy: hierarchy:19423
optionLabel: hierarchy:language_text
default: @currentUser.nodeId
root: @currentUser.nodeId

}
}
```

As with any other applied filters, Fixed filters will be shown in the filter summary.

7.5.2.3. Fixed Filter

A Fixed filter is a filter configured by the dashboard designer to be always applied; it can't be switched off by the end user. For this purpose it is best to use a singleselect filter, as it must then always have one option selected. Below is a code example for a fixed filter:

```
layoutArea toolbar {
filter singleselect filterIdentifier {
label: "Reporting period"
global: false // true by default, if set to true, this filter would be
applied
globally to every widget on a page when selected
// optional
option radio o1 {
selected: true // option selected by default, if you want to use this
filter
label: "Rolling Year"
value: InYear(survey:interview_start, -1, 0)
previous: InYear(survey:interview_start, -2, -1)
}
option radio o2 {
label: "Rolling Quarter"
value: InQuarter(survey:interview_start, -1, 0)
previous: InQuarter(survey:interview_start, -2, -1)
}
option radio o3 {
label: "Rolling Month"
value: InMonth(survey:interview_start, -1, 0)
previous: InMonth(survey:interview_start, -2, -1)
}
}
}
```

As with any other applied filters, Fixed filters will be shown in the filter summary.

7.5.2.4. Filter Properties

Below are the properties you can set for a filter:

Property	Purpose	Data Type	Example/Default Value	Remarks
layoutArea	defines the page area where the filter will be placed	string	toolbar	
multiselect	defines a multiselection filter	string		
singleselect	defines a single selection filter	string		

Property	Purpose	Data Type	Example/Default Value	Remarks
optionsFrom	defines the source of the filter options	path		
option	defines the selection control type	string	radio / checkbox	
selectedOption	defines the option selected by default			
label	defines the filter / filter option label	string		
value	defines the value which the filter option is based on	expression	value:accounts:TotalAccountValue > 200000	
previous_survey				
previous_hc				

```
layoutArea toolbar {
  filter multiselect accountOwenr {
    label: "Account Owner"
    optionsFrom: accounts:AccountOwner
```

Figure 60 Example of a filter

7.5.3. End-User Selectable Filters

Several types of end-user selectable filters are available in Studio:

- **Multiselect** - allows you to choose several options.
- **Singleselect** - allows you to choose a single option.

Filters are usually defined in filter sets, to be displayed on a single toolbar located in a defined page layout area.

```
  contactLogo: "/isa/BDJPFRDMEYBPBKLVADAYFQCDAVIOEQJR/mch/53633418-5037-4CEB
  currentPeriodFilter: survey:interview_start >= 2016-06-22
  previousPeriodFilter: survey:interview_start < 2016-06-22
  }
layoutArea toolbar {
  filter multiselect {
    optionsFrom: survey:NPSSegment
  }
  filter multiselect {
    optionsFrom: survey:AccountName     ///NPSSegment
  }
  filter multiselect {
    optionsFrom: cases:lk_7
  }
  filter multiselect {
    label: "Account Rating"
    option checkbox {
      label:"Go545451d1"
        value:accounts:TotalAccountValue > 200000
      }
      option checkbox {
        label:"Silver"
        value: accounts:TotalAccountValue >99999 AND accounts:TotalAccountVa
      }
      option checkbox {
        label:"Bronze"
        value: accounts:TotalAccountValue < 100000
      }
```

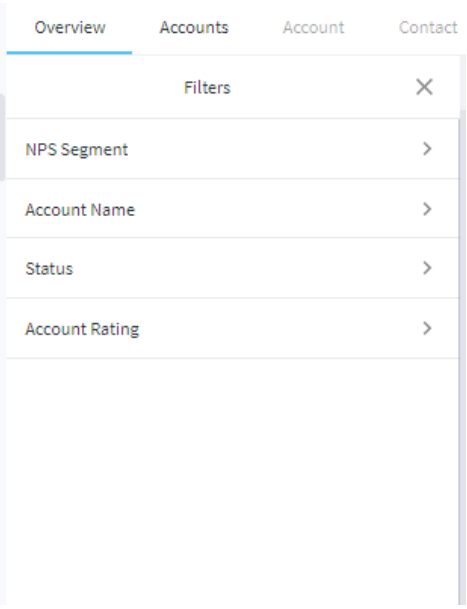


Figure 61 Example of multiple filters on a Filter toolbar

Code example:

```
layoutArea toolbar {
filter multiselect {
optionsFrom: survey:NPSSegment
}
filter multiselect {
optionsFrom: survey:AccountName ///NPSSegment
}
filter multiselect {
optionsFrom: cases:lk_7
}
filter multiselect {
label: "Account Rating"
option checkbox {
label:"Go54545ld1"
value:accounts:TotalAccountValue > 200000
}
option checkbox {
label:"Silver"
value: accounts:TotalAccountValue >99999 AND accounts:TotalAccountValue
<199999
}
option checkbox {
label:"Bronze"
value: accounts:TotalAccountValue < 100000
}
}
```

7.5.4. Using Selected Filter Values in the Report

If you name your filter (give it an identifier), you can apply the selected filter value as the filter option in a widget. For example you can use it in a column when calculating the metric for the current reporting period, with an indication if the value has changed since the previous reporting period.

Available properties:

- @filterIdentifier.selectedOption.label - the label given to the currently selected option.
- @filterIdentifier.selectedOption.value - the value of the currently selected option.
- @filterIdentifier.selectedOption.previous - an optional property, available if you specified a previous property in your filter definition with a data expression that would calculate the value for the previous reporting period (see the code example below).

```
layoutArea toolbar {
filter singleselect filterIdentifier {
value: InYear(survey:interview_start, -1, 0)
previous: InYear(survey:interview_start, -2, -1)
}
option radio o2 {
label: "Rolling Quarter"
value: InQuarter(survey:interview_start, -1, 0)
previous: InQuarter(survey:interview_start, -2, -1)
}
option radio o3 {
label: "Rolling Month"
value: InMonth(survey:interview_start, -1, 0)
previous: InMonth(survey:interview_start, -2, -1)
}
}
}
```

```
Filters.md 9/12/2018
5 / 6
label: "Reporting period"
global: false // if selected filter should be applied globally
option radio o1 {
selected: true // option selected by default
label: "Rolling Year"
value: InYear(survey:interview_start, -1, 0)
previous: InYear(survey:interview_start, -2, -1)
}
option radio o2 {
label: "Rolling Quarter"
value: InQuarter(survey:interview_start, -1, 0)
previous: InQuarter(survey:interview_start, -2, -1)
}
option radio o3 {
label: "Rolling Month"
value: InMonth(survey:interview_start, -1, 0)
previous: InMonth(survey:interview_start, -2, -1)
}
}
}
page "Filter summary" {
// see what's selected
widget title {
layout column {
tile value {
value: @filterIdentifier.selectedOption.label
}
}
}
// you can pass the filter to data expression
widget accountList{
label: @filterIdentifier.selectedOption.label // dynamically change the
label
of the widget
size: large
table: accounts:
sortColumn: accountName
sortOrder: ascending
navigateTo: "Account"
hierarchy: accounts:ParentAccountID
view metricWithChange metrics {
backgroundColorFormatter: backgroundColor
valueColorFormatter: valueColor
fontSize:medium
roundCorners:true
}
column value accountName {
value: accounts:AccountName
}
column metric LTR {
label: "LTR"
value: average(score(survey:Q1), @filterIdentifier.selectedOption.value)
Filters.md 9/12/2018
6 / 6
previous: average(score(survey:Q1),
@filterIdentifier.selectedOption.previous)
target: 9
format: formatterLTR
align: center
view: metrics
}
```

```
}
}
```

7.5.5. Using Filters in the Report

Note: Filters are active only for the current session. If the session is closed for any reason then the filter will be removed from the report.

When you have added filters to your report, the report will display the **Filter** icon in the upper-left corner of the report page. Click this to open the Filters panel down the left side of the report.

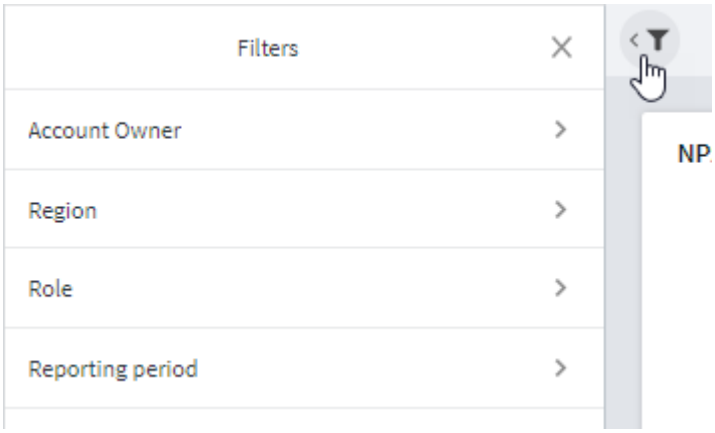


Figure 62 Example of the Filters panel

The filters panel lists all the filters that are configured in the report (see Filters on page 62 for more information). To set a filter for the report:

1. Click the **>** icon to open the filter.

   The filter criteria for that filter are presented.

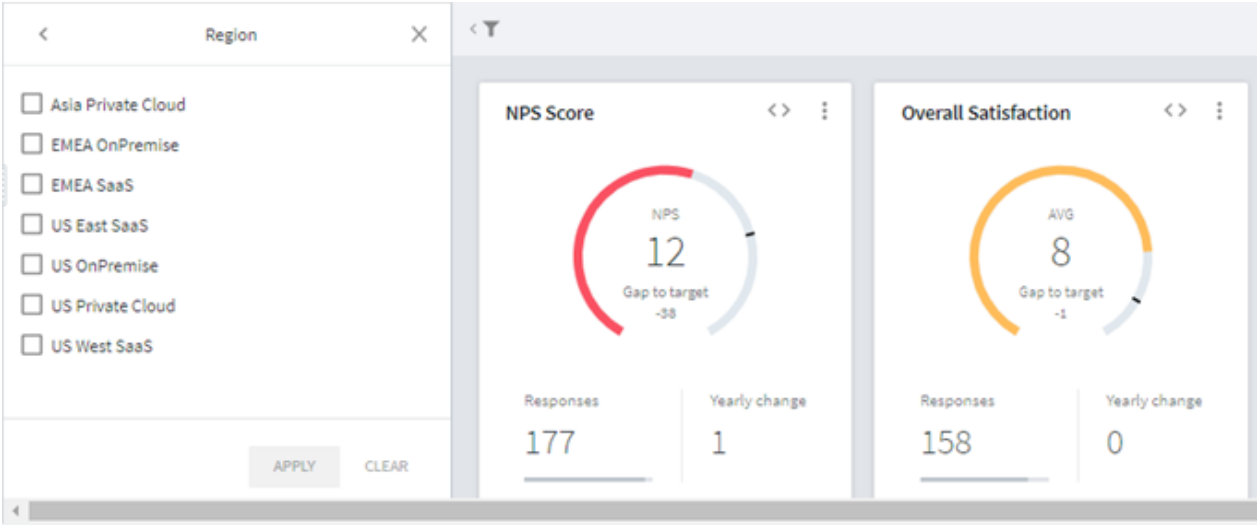


Figure 63 Example of a Region filter opened for selection

2. Select the criteria you wish to filter the data by.

The data is updated as you make your selections, so you can see immediately the results of the filter.

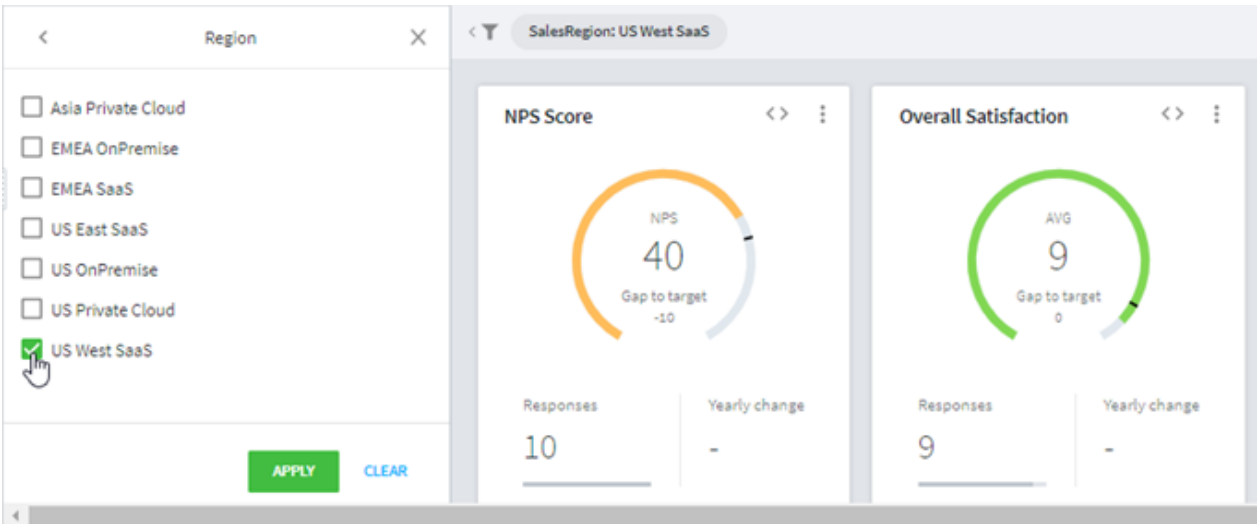


Figure 64 Selecting the filter criteria

3. When you have completed your selections, click **Apply** towards the bottom of the Filter panel.

   The filter that is applied, and the criteria selected, are displayed in the Filters panel and beside the **Filter** icon above the report.

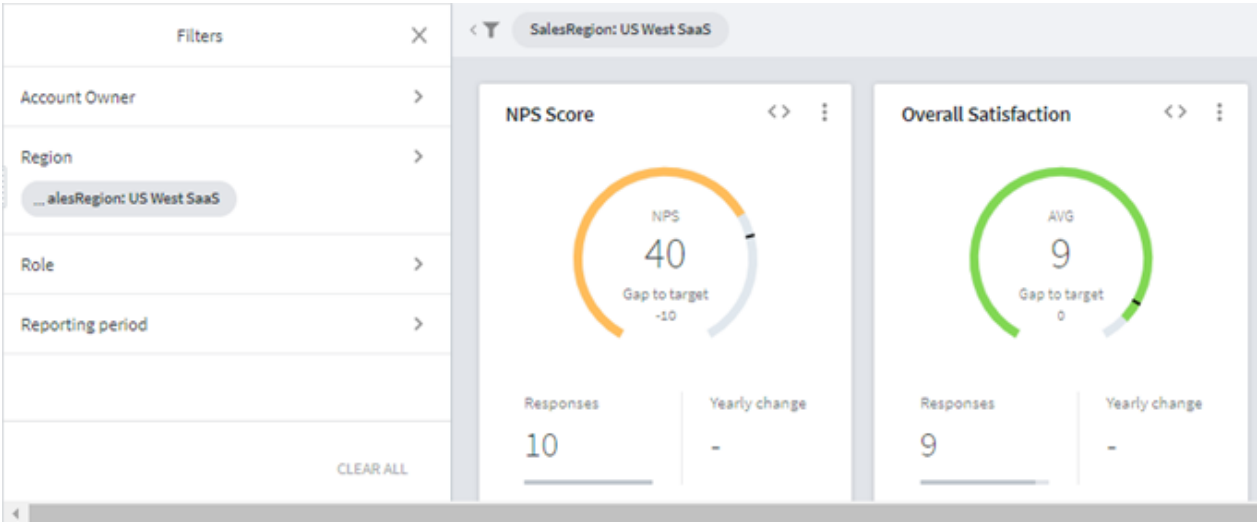


Figure 65 The filter criteria applied

- To close the Filters panel, click the **X** button in the panel toolbar.
- To reopen the panel at a particular filter, click the appropriate filter criteria beside the **Filter** icon.
- You can deactivate a filter, that is switch it off without actually deleting it. This allows you to reactivate the same filter later without having to rebuild it. To do this, in the Filters panel hover over the activated filter and select the **Deactivate** icon .

Note: When a user applies a filter to a report, it will be active only for the current session. If the session is closed for any reason (the user closes the report, the session times out etc.) then the filter will be removed from the report.

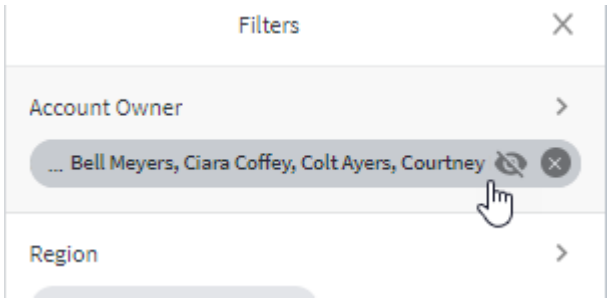


Figure 66 Deactivating a filter

- To clear one filter, in the Filters panel hover over the activated filter and select the **Clear filter** icon .
- To clear all the filters, click **Clear All** towards the lower-right corner of the Filters panel.

7.6. Drilldown

The drillDown block defines a hierarchical structure of different tables that can be represented as a tree. This structure can be used in the filter `filter drillDown` on a report level or in the special `drillDown` column of the list widget.

```
drillDown "myRoute" {
level distinct {
table: accounts:
value: accounts:SalesRegion
}
level distinct {
table: accounts:
value: accounts:AccountOwner
}
level hierarchy {
hierarchy: accounts:ParentAccountID
value: accounts:AccountName
}
level directory {
table: contacts:
value: contacts:label
}
level distinct {
table: survey:
value: survey:NPSSegment
}
}
```

Figure 67 Example of a drillDown definition

The drillDown block contains a set of levels, that are in effect references to table fields and columns. There are several types of levels, and an arbitrary number of levels can be used in any order. The level types are as follows:

- **distinct -** this level type is used when the referenced table column may contain repeated values, which will be displayed as a single tree node in the current level's node list. This level has the following properties:
  - **table -** the name of the referenced table.
  - **value -** the vpath to the referenced table field.

- **hierarchy -** this level type is used to refer to a table that defines a self-referenced (unblanced) hierarchy. This level has the following properties:
    - **hierarchy -** the vpath to the hierarchy-defining column (parent node id) of the self-referenced (unbalanced) hierarchy table.
    - **value -** the vpath to the column used to build the current level's node list.
- **directory -** this level type is used when the referenced table column does not contain repeated values. This level has the following properties:
    - **table -** the name of the referenced table.
    - **value -** the vpath to the referenced table field.

7.6.1. Filter Drilldown

This filter uses the hierarchical structure defined in the `drillDown` block. To be used on a report level it must be defined within a `layout area`. An example of the code is :

```
layoutArea toolbar {
filter drillDown {
drillDown: myRoute
label: "Drill down"
}
}
```

Figure 68 Example of a drillDown filter definition

where:

- **drillDown -** the ID of a drilldown filter defined in the drilldown block.
- **label -** the label that is to be displayed for the filter. Type in the required text.

7.6.2. dataBasedTables

These virtual tables are used in the "data-driven hierarchy" scenario.

A date-driven hierarchy is based on the data contained in the table's records. This hierarchy automatically creates nodes and values based on the table's data. While it is easy to create and requires no maintenance for the structure, it does require that the data in the responses is consistent and ‘clean’ in order to build a usable hierarchy. The survey-based hierarchy can be built from answers to questions or background variables. With this type of hierarchy the data remains where it is; if you move, the data remains. So if you used to work for Adam but you now work for Bill, all the data that you collected while working for Adam will remain with Adam and will not be available to Bill (or you).

Below is an example of a data-driven hierarchy. Let us assume the original survey is as follows ('region' and 'segment' are Singles with values at the time of response):

AccountID	region	segment
a1	r1	s1
a1	r2	s1
a1	r1	s2

After "virtual conversion" we get:

- a virtual table for regions:

region
r1
r2

- a virtual table for segments:

segment
s1
s2

and the original survey table is virtually a child table of these tables.

The dataBasedTables are defined as follows:

```
dataBasedTables { //maybe call it dataBasedHierarchy as it creates not
only tables but also relations between them
origin: p12345.response //table from which virtual tables are constructed
foreignKey: @cr.reportingPeriod //optional filter to be applied to the
original data

level region { //region will be a name of table
groupBy: expression
nameAs: label //name of variable created in virtual table. could be
"label" by default
}
level accountOwner { //accountOwner will be a name of table
groupBy: expression
nameAs: label //name of variable created in virtual table. could be
"label" by default
}
```

7.7. Role Based Access

You can hide pages and/or widgets based on a user's properties (claims). These can be either claims the user is given by the identity service, or custom claims uploaded as a table in the hub. If you upload the claims as a hub table, you can configure the report to use this table to extend the claims for a user.

In the config hub block, we can add **userProperty** of type **claim**.

```
// additional Role based access code
userProperty claim myRole {
  joinKey: accessRules:Username
  value: accessRules:Role
}

userProperty claim myFullName {
  joinKey: accessRules:Username
  value: accessRules:UnderscoredName
}
userProperty claim myRegion{
  joinKey: accessRules:Username
  value: accessRules:SalesRegion
}
userProperty claim myRegionName{
  joinKey: accessRules:Username
  value: accessRules:Region
}
```

Figure 69 Example of the code to implement role-based access

- Claim - adds another property on the @currentUser object that can then be referred to elsewhere in the DSL

To show a page or a widget only for people with certain claims, add an access rules block to the item.

```
page "Welcome" {
  widget markdown {
    access rules { // this widget will be visible to user with myRole="Manager" AND myRegionName="US_VoE"
      rule claim {
      name: "myRole"
      value: "AccountOwner"
      }
      rule claim {
      name: "myRegionName"
      value: "US_VoE"
      }
    }
    markdown: "Now you see me"
  }
}
```

Figure 70 Example of the code to allow role-based access

In the event more than one rule is included, logical AND is used so all the rules must be true for the item to be shown. In the example above, only a user with both myRole="AccountOwner" AND myRegionName="US_VoE" will be able to see the markdown widget on the Welcome page.

7.8. Widget Configuration

The block of code to configure a widget must be located within a Page block. For example, the KPI widget with the title NPS Score is located in the Overview page.

```
page "Overview" {
  widget kpi {
    label: "NPS Score"
    size: small
    tile kpi {
      label:"NPS"
      value: NPS(survey:Q1)*100
      target: @cp.npsTarget
      min: -100
      max: 100
      format:formatterLTR
      targetFormat:formatterLTR
      gaugeColorFormat:kpiColorFormatter  // valueColor
      tile value {
        label: "Responses"
        value: count(survey:Q1,@currentPeriodFilter)
        max: count(survey:responseid, @cp.currentPeriodFilter)
        format: integer
      }
      tile value {
        label: "Yearly change"
        value: average(score(survey:Q1),@cp.currentPeriodFilter) -average(score(survey:Q1),@cp.previo
        format:formatterLTR
      }
    }
  }
```

Figure 71 Example of a widget configured within a page block

7.9. Layout Area

A page in studio consists of four areas: header, toolbar, main area with widgets, and footer. The layoutArea code allows the dashboard designer to define the appearance of the various parts of the page. Note that the "header", "toolbar" and "footer" areas are predefined; all other widgets defined in the page will be rendered in the main area. See the code examples below.

- The layoutArea containers in dsl can be defined either inside a page, or globally, or both. Generally, the containers may contain widgets, though some constraints may be imposed by the schema.
- The layoutArea containers can be defined either by subType (for example layoutArea toolbar {}) or by identifier (for example layoutArea "toolbar" {}).
- The toolbar panel will be present only if there is a toolbar layout area defined in dsl. All filters must be defined inside the toolbar panel, not in the global scope.
- If the same layoutAreas are defined both globally and inside a page, then for each layout area for each page the global items are rendered first and the page-specific items are rendered afterwards. Note that these will be concatenated, not replaced or merged.

```
config hub {...}
config report {...}
layoutArea header {
    widget global_header_widget {
      // this widget will be rendered in the header
    }
}
layoutArea toolbar {
    filter multiselect {
      ...
    }
}
page "Page" {
    layoutArea header {
        widget header_widget {
            // this widget will be rendered in the header along with global_header_widget
        }
    }
    widget widget_1 {
        // this widget will be rendered in the main area
    }
    widget widget_2 {
        // this widget will be rendered in the main area
    }
}
```

Figure 72 General example of the layoutArea code

8. Time Ranges

Time ranges allow the end user to filter the report content and track and analyze changes of critical metrics in time, by comparing the current period values against the previous period values. You can define time ranges by using absolute dates or relative dates based on time periods.

Studio supports the following time range units:

- Year
- Quarter
- Month
- Week
- Day

Time ranges can be defined on a report level and used in filter definitions.

The syntax for expressions with time period absolute dates is as follows:

```
InPeriod(date, n_start, n_end, ref_date)
```

where

- Period is a time period, for example, `inMonth`;
- date is a time variable being evaluated against the range,
- n_start is the number of time periods in relation to the reference date that define the start date of the range;
- n_end is the number of time periods in relation to the reference date that define the end date of the range;
- ref_date is the reference date.

The syntax for expressions with time period absolute dates is as follows:

```
Between(date, start_date, end_date)
```

The expressions evaluate to true if the specified `date` is within the specified range of periods relative to the specified start date. The start date defaults to current date producing a rolling period.

```
currentPeriod: between(survey:interview_start, 2016-07-01, 2016-08-31)
previousPeriod: between(survey:interview_start, 2016-05-01, 2016-06-30)
// In period with explicit start date
// currentPeriod: InMonth(survey:interview_start, -1, 0, 2016-08-01)
// previousPeriod: InMonth(survey:interview_start, -3, -2, 2016-08-01)
// rolling period
//currentPeriod: InMonth(survey:interview_start, -21, -20)
//previousPeriod: InMonth(survey:interview_start, -23, -22)
// same months 2 years ago
//currentPeriod: InMonth(survey:interview_start, 0, 3)
//previousPeriod: InMonth(AddYear(survey:interview_start, 2), 0, 3)
// Period to current date
//currentPeriod: YearToDate(survey:interview_start)
//previousPeriod: InYear(survey:interview_start, -2, -1)
```

9. Widget Types

Studio includes a number of different widgets that you can add to your report. The widgets described in this user guide are available in all Studio-based solutions. Note that additional widgets may be available in specific solutions - if so then these will be described in the user guide applicable to that particular solution.

The widgets generally available are listed in the following sections.

Note: Code examples for configuring the widgets are given when available. Be aware that these are examples only, and will probably not give the results shown in the images on the associated pages - some editing must be expected.

To allow these code examples to be more easily copied directly from this documentation, indentation is not applied here. You are advised to add appropriate indentation to your code to simplify later editing and understanding of its use.

To reduce the requirement for updating, the properties available for the widgets are listed in the Studio Reference Guide.

Important
You are strongly recommended to add comments into the CDL at every opportunity **(see Comments in CDL on page 16 for more information).**

9.1. Account List Widget

The Account List widget is a complex table widget. It is intended to give a Report Viewer, for example a business user such as Account Owner (AO), a list of accounts with all relevant metrics and records: for example VoC survey results scores to number of cases, number of responses etc.

This widgets consists of:

- Widget Title
- Columns
- Widget menu

The table responds to report filter selection and has responsive view.

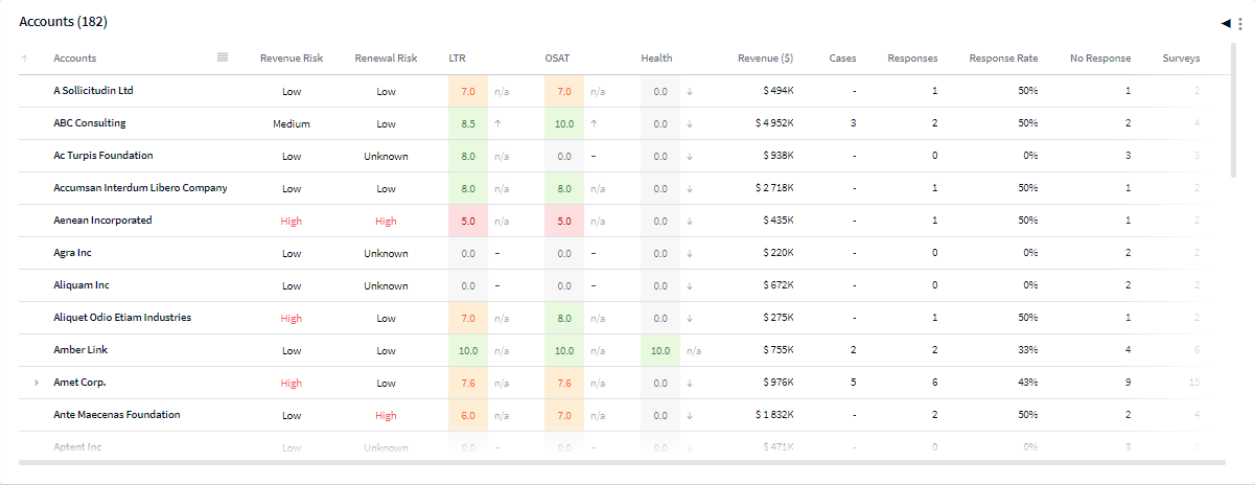


Figure 73 Example of the Account List widget

9.1.1. Account List Code Example

Below is an example of the code used to create and set up an Account widget:

```
widget accountList{
label: "Accounts"
size: large
table: accounts:
sortColumn: accountName
sortOrder: ascending
navigateTo: "Account"
hierarchy: accounts:ParentAccountID
// views
view metricWithChange metrics {
backgroundColorFormatter: backgroundColor
valueColorFormatter: valueColor
fontSize:small
// roundCorners:true
}
view metric risk {
backgroundColorFormatter: riskTextBgColorFormatter
valueColorFormatter: riskTextColorFormatter
fontSize:small
// roundCorners:true
}

// columns
column hierarchy accountName {
label: "Accounts"
value: accounts:AccountName
rowHeader: true
}

column metric revenueRisk {
label: "Revenue Risk"
value: @cp.riskValue
target: 1
format: riskStringFormatter
valueColorFormatter: riskBgColorFormatter
//view: iconSmall
view: risk
}
column metric renewalRisk {
label: "Renewal Risk"
value: @cp.renewalRiskValue
target: 1
format: riskStringFormatter
//view: iconSmall
view: risk
}
column metric LTR {
label: "LTR"
value: average(score(survey:Q1),@cp.currentPeriodFilter)
previous: average(score(survey:Q1), @cp.previousPeriodFilter)
target:@cp.ltrTarget
format: formatterLTR
align: left
view:metrics
}
column metric osat {
label: "OSAT"
value: average(score(survey:Q4),@cp.currentPeriodFilter)
previous: average(score(survey:Q4), @cp.previousPeriodFilter)
```

```
format: formatterLTR
view:metrics
target:@cp.osatTarget
align:left
}
column metric health11 {
label: "Health"
value : average(score(healthCheck:Renew), @cp.currentPeriod)
previous: average(score(healthCheck:Renew), @cp.previousPeriod)
target: @cp.healthTarget
format: formatterLTR
view: metrics
align: left
}
column value total {
label: "Revenue ($)"
value:accounts:TotalAccountValue
format: currency
}
column value case1 {
label: "Cases"
value:@cp.casesValue
format: customEmpty
}
column value responses {
label: "Responses"
value: @cp.completeSurv
align: right
}
column value rate {
label: "Response Rate"
value: @cp.rateValue
format: formatterRR
}
column value noResp {
label: "No Response"
align: right
value: COUNT(survey:response)-@cp.completeSurv
//COUNT(survey:response,survey:smtpstatus="Sent")
}
column value survCount {
label: "Surveys"
value: count(survey:response)
align: right
}
```

9.2. Bar Chart Widget

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally. A bar chart shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

In a stacked bar chart, several sets of data can be represented by one bar or stack. For each row, individual data elements and the sum of all the data elements can be displayed.

Bar charts support Single choice, Numeric, Date and Grid variables, with some limitations. They also support NPSSegment emulated functions.

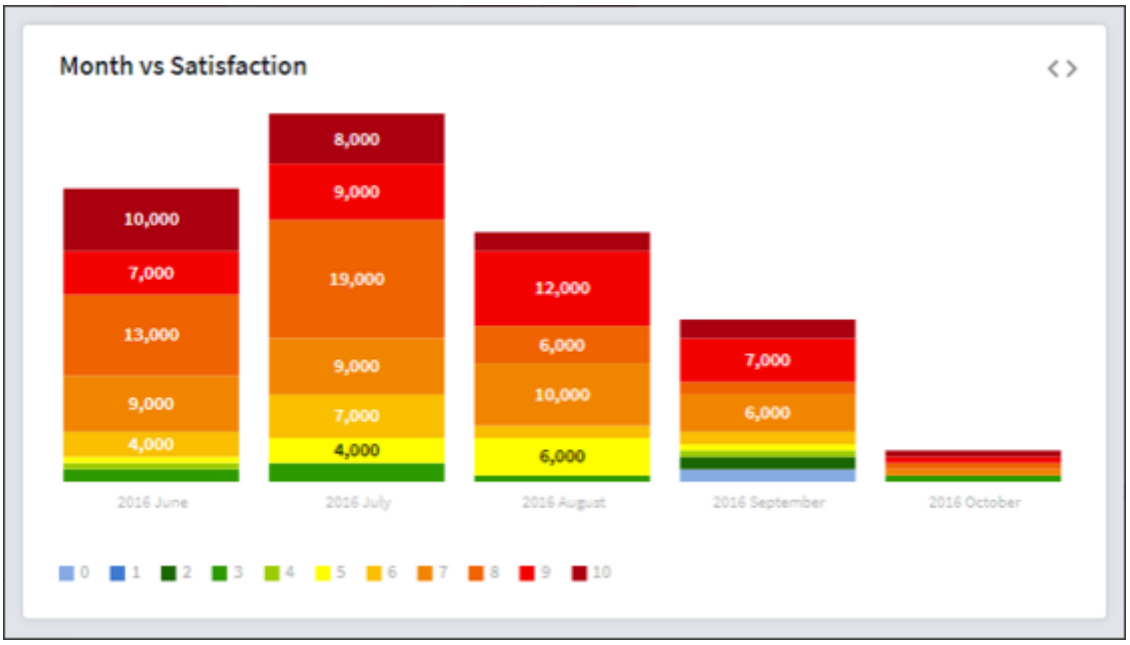


Figure 74 Example of a bar chart

9.2.1. Bar Chart Code Example

Below is an example of code to configure a bar chart widget.

```
widget barChart {
label: "Month vs Satisfaction"
size: medium
palette:
"#86ABE2","#4079D0","#1B6600","#2D9900","#9CCB00","#FEFE00","#F9BF00","#F
18500","#EF6300","#F30000","#AA0010","#C0C0C0"
navigateTo: "Responses"
category date {
value: survey:interview_start
breakdownBy: calendarMonth
format: dateFormat
start: "2018-06-01"
end: "-1 years"
removeEmpty: true
}
series cut {
value: survey:Q1
}
value: count(survey:response)
format: floatNumber
}
```

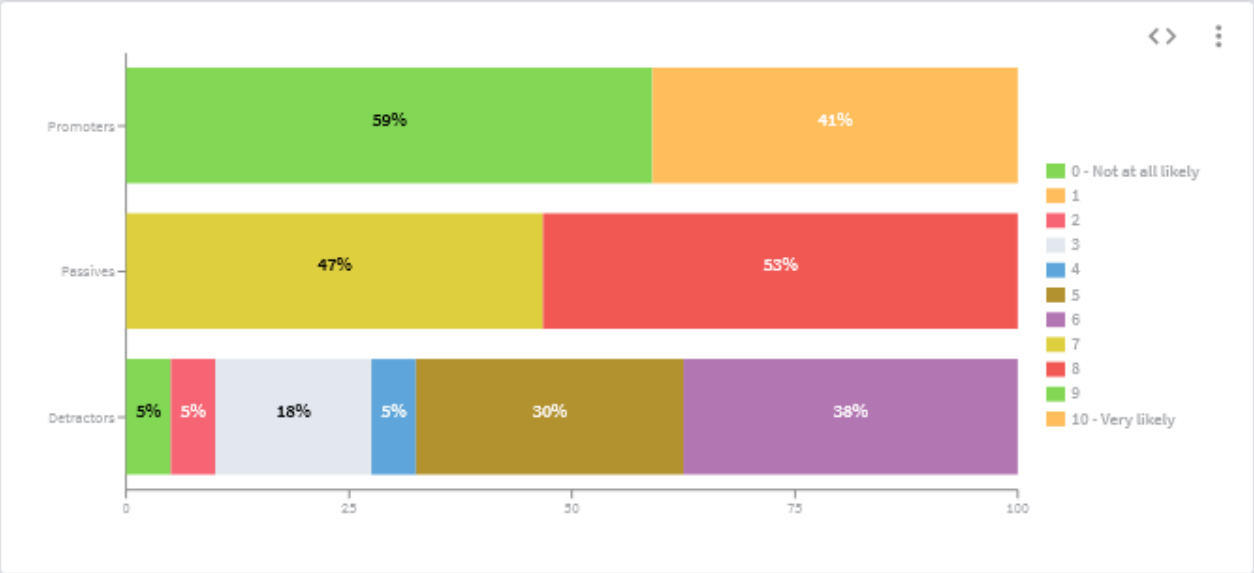
9.3. Chart Widget

This is a generic widget that enables you to create a number of different types of charts. It supports a wide range of visualization options for area, trendline, pie and bar charts, and combinations of these.

The widget has a vast number of configurations options:

- Legend placement

- Navigation
- Horizontal/vertical bar orientation
- Gridlines
- Stacked bars
- Rolling average, and more.



Example of a chart created by the Chart widget

9.3.1. Chart Widget Code Example

Below is an example of code to configure a chart widget.

```
widget chart {
size,
label,
access rules {
...
},
infobox {
...
},
filter expression {
...
},
ignoreFilters,

navigateTo,
legend,
animation,
layout,
removeEmptyCategories,
removeEmptySeries,
palette,
scroll,
chart ...
series ...
category ...
```

```
base ...
}
```

9.3.2. Chart Code Block

This block defines the type of chart that is to be created. Note that some properties can be chart type dependent. The chart definition defined within the widget scope is applied for all series, though for some series the chart type can be redefined (for example to construct combined charts). In both cases the same syntax is used.

9.3.2.1. Chart Code Examples

Below are examples of the code to configure the chart type:

```
chart bar {
stacked,
stackId,
barColorFormat,
axis ...
}

chart area {
stacked,
stackId,
lineType,
lineWidth,
dotSize,
dotColorFormat,
connectNulls,
axis ...
}

chart line {
lineType,
lineWidth,
dotSize,
dotColorFormat,
connectNulls,
axis ...
}

chart pie {
}
```

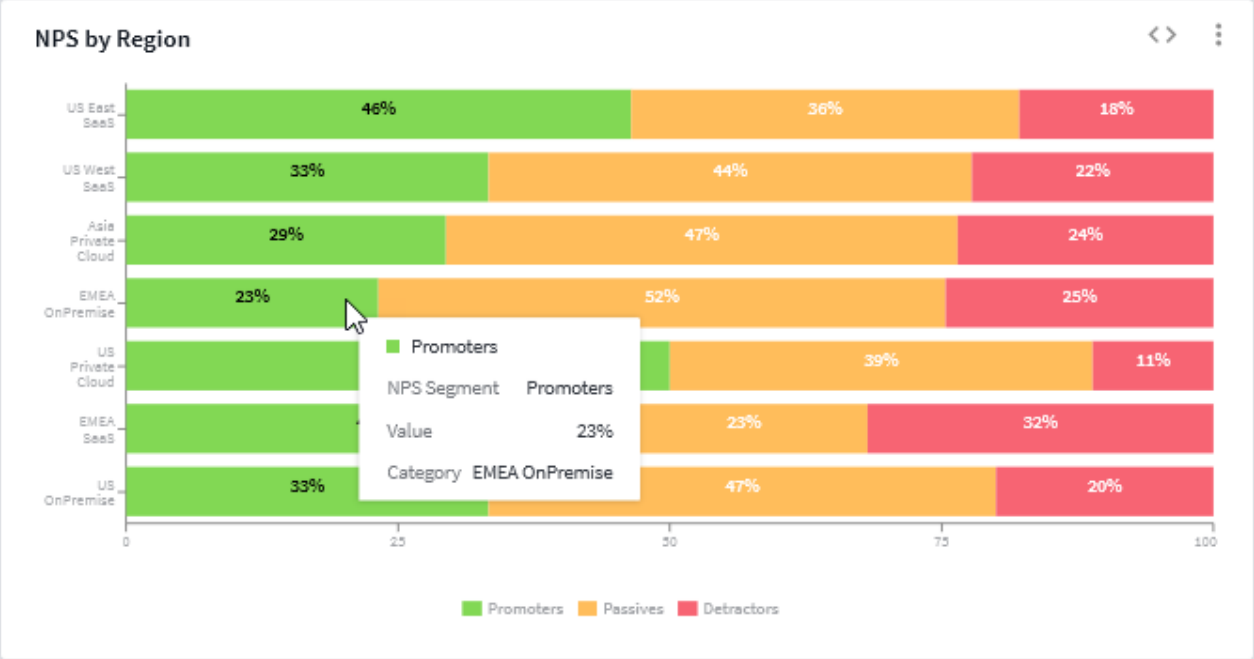


Figure 75 Example of a bar chart

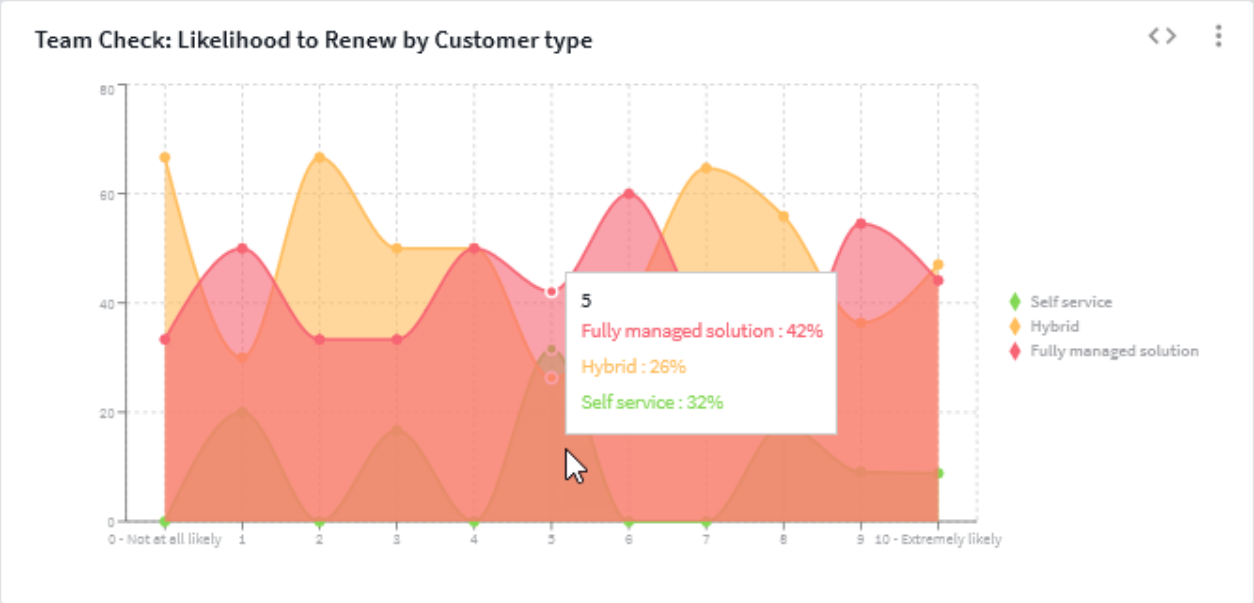


Figure 76 Example of an area chart

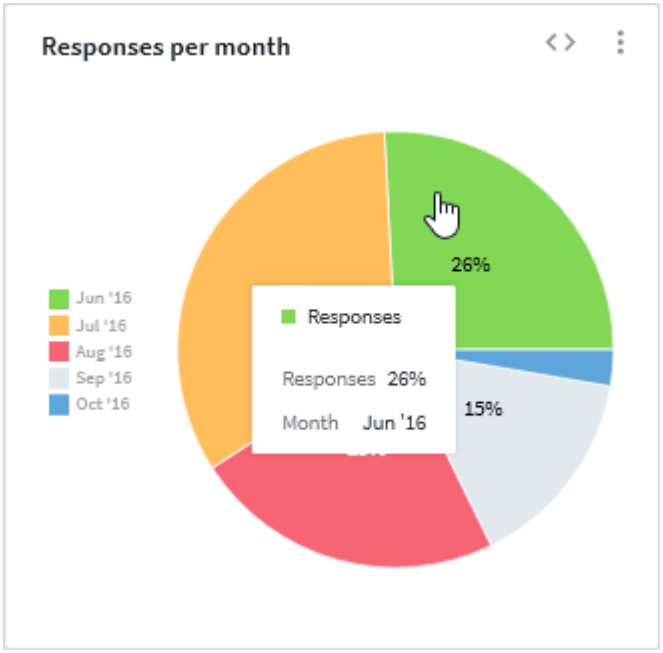


Figure 77 Example of a pie chart

9.3.3. Axis Code Block

This code block defines the axis properties that could be chart type dependent.

9.3.3.1. Axis Code Example

Below are examples of the code to configure the chart axes:

```
axis category {
hide,
interval,
axisLine,
tickLine
}

axis number {
hide,
axisLine,
tickLine,
minValue,
maxValue,
isSecondary
}
```

9.3.4. Series Code Block

This block defines the series properties that could be chart type dependent.

9.3.4.1. Series Code Example

Below is an example of the code to configure a chart series:

```
series {
value,
label,
format,
percent,
```

```
isSecondary,
breakdownBy...,
chart ...
}
```

9.3.5. BreakdownBy Code Block

The BreakdownBy code block defines how the data presented in a stacked bar chart is to be divided into its various groups. In the figure below the chart is broken down by NPS segments.

Figure 78 Example of a stacked bar chart broken down by NPS segments

And below is the code that was used to create the chart:

```
widget chart {
    label: "NPS by Region with Satisfaction"
    size: medium
    navigateTo: "p3"
    legend: rightMiddle
    experimentalTooltip: true
    chart bar {
      stacked: true
      axis number {
        minValue: -10
        maxValue: 8.5
        isSecondary: true
      }
    }
    category cut {
      value: survey:SalesRegion
    }
    series {
      value: count(survey:)
      breakdownBy cut {
        value: survey:NPSSegment
      }
    }
    series #st {
      value: avg(score(survey:Q4))
```

Figure 79 The code used to create the chart

9.3.5.1. BreakdownBy Code Example

Below are examples of the code to configure how the chart is to be broken down:

```
breakdownBy cut {
value,
label,
format,
}
breakdownBy cutByMulti {
value,
label,
format,
}
breakdownBy date {
value,
label,
format,
breakdownBy,
start,
end
}
breakdownBy overlappingdate {
value,
label,
format,
breakdownBy,
start,
end,
startShift,
endShift
}
```

9.3.6. Category Code Block

The category code block defines how the chart is to divided into, for example for a bar chart, its separate columns. For example, in the figure below the chart is divided into columns for the various sales regions.

Figure 80 Example of a stacked bar chart categorized by sales region

And below is the code that was used to create the chart:

```
widget chart {
    label: "NPS by Region with Satisfaction"
    size: medium
    navigateTo: "p3"
    legend: rightMiddle
    experimentalTooltip: true
    chart bar {
      stacked: true
      axis number {
        minValue: -10
        maxValue: 8.5
        isSecondary: true
      }
    }
    category cut {
      value: survey:SalesRegion
    }
    series {
      value: count(survey:)
      breakdownBy cut {
        value: survey:NPSSegment
      }
    }
    series #st {
      value: avg(score(survey:Q4))
```

Figure 81 The code used to create the chart

9.3.6.1. Category Code Examples

Below are examples of the code to configure the chart categories:

```
category list {
value,
label,
format,
hierarchy,
table,
skip,
take,
parentKey,
sortOrder,
sortBy,
index,
emptyFirst,
takeTop,
takeBottom,
base ...
}
category cut {
value,
label,
format,
sortOrder,
sortBy,
index,
emptyFirst,
takeTop,
takeBottom,
base ...
}
category cutByMulti {
value,
label,
format,
sortOrder,
sortBy,
index,
emptyFirst,
takeTop,
takeBottom,
base ...
}
category date {
value,
label,
format,
breakdownBy,
start,
end,
sortOrder,
sortBy,
index,
emptyFirst,
takeTop,
takeBottom,
base ...
}
category overlappingDate {
value,
label,
format,
```

```
breakdownBy,
start,
end,
startShift,
endShift,
sortOrder,
sortBy,
index,
emptyFirst,
takeTop,
takeBottom,
base ...
}
```

9.3.7. Base Code Block

The Base code block allows you to include an additional value on a chart axis, for example the number of respondents who replied to a question.

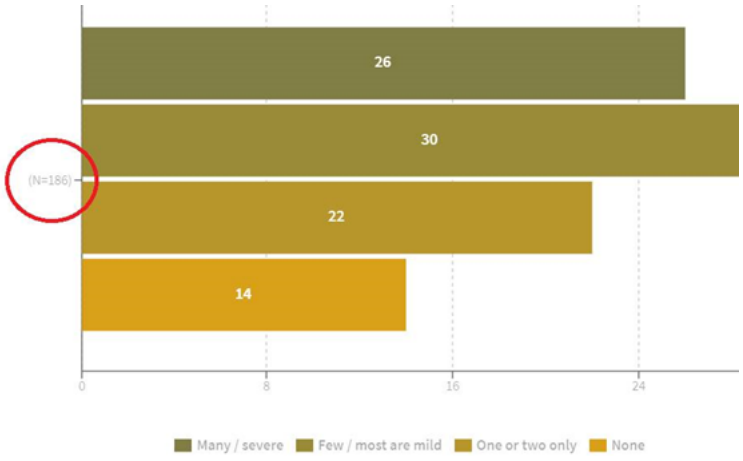


Figure 82 Example of the Base in use

9.3.7.1. Base Code Example

Example of the code to configure the chart base:

```
base {
value,
format,
hide
}
```

9.3.8. Additional Examples of Code

This section presents some additional examples of code that can be used to set up a Chart widget to present data in various ways. Note that as the functionality, look and feel of the resulting chart will depend on the data available and the other settings and selections in your Studio report, editing of this code must be expected.

9.3.8.1. Bar Chart Code Example

The following is an example of the code used to set up a Chart widget as a bar chart. Note that as the functionality, look and feel of the resulting chart will depend on the data available and the other settings and selections in your Studio report, editing of this code must be expected.

```
widget chart {
label: "BarChart"
```

```
size: large
legend: bottomLeft
animation: true
chart bar {
stacked: true
axis category {
axisLine: false
tickLine: false
}
axis number {
hide: true
}
}
series {
value: count(survey:responseId)
label: "Percent:"
percent: true
format: percentNumber
breakdownBy cut {
value: survey:Q1
label: "Satisfaction"
}
}
category cut {
value: survey:Industry
format: textFormat
label: "Industry"
}
}
```

9.3.8.2. Trend Chart Code Example

The following is an example of the code used to set up a Chart widget as a trend chart. Note that the functionality, look and feel of the resulting chart will depend on the data available and the other settings and selections in your Studio report, so you must expect to have to edit this code.

```
widget chart {
label: "Trend"
size: medium
navigateTo: "Responses"
palette: @cr.palette
legend: bottomCenter
removeEmptyCategories: true
experimentalTooltip: true
chart line {
lineType: monotone
dot: false
axis number {
minValue: -1
maxValue: 70
}
}
series {
label: "Responses"
value: COUNT(survey:responseid)
format: floatNumber
}
series {
label: "Complete"
value: COUNT(survey:responseid, survey:status = "Complete")
format: floatNumber
chart line {
lineType: basis
dot: false
```

```
}
}
series {
label: "No Response"
value: COUNT(survey:responseid) - COUNT(survey:responseid, survey:status
= "Complete")
format: floatNumber
chart line {
lineType: step
dot: true
}
}
category date {
value: survey:interview_start
breakdownBy: calendarMonth
label: "Date"
format: dateFormat
}
}
```

9.3.8.3. Pie Chart Code Example

The following is an example of the code used to set up a Chart widget as a pie chart. Note that the functionality, look and feel of the resulting chart will depend on the data available and the other settings and selections in your Studio report, so you must expect to have to edit this code.

```
widget chart {
label: "Pie"
navigateTo: "Responses"
legend: leftMiddle
chart pie {
}
series {
value: count(survey:responseId)
label: "Responses"
percent: true
}
category date {
value: survey:interview_start
breakdownBy: calendarMonth
format: dateFormat
label: "Month"
start: "2016-01-01"
end: "2017-01-01"
}
removeEmptyCategories: true
}
```

9.3.8.4. Composite Chart Code Example

The following is an example of the code used to set up a Chart widget as a composite chart. Note that the functionality, look and feel of the resulting chart will depend on the data available and the other settings and selections in your Studio report, so you must expect to have to edit this code.

```
widget chart {
size: large
legend: bottomLeft
layout: vertical
scroll: enabled
experimentalTooltip: true
removeEmptyCategories: true
series {
value: count(survey:responseId)
//percent: true
```

```
breakdownBy cut {
value: survey:Q1
}
chart bar {
stacked: true
}
}
series {
value: 50
label: "Target"
chart line {
lineType: monotone
}
}
series {
value: avg(survey:TotalAccountValue)
//percent: true
format: floatNumber
isSecondary: true
label: "Account value"
chart area {
//stacked: true
}
}
category date {
label: "c1"
value: survey:interview_start
breakdownBy: calendarMonth
format: dateFormat
sortOrder: descending
takeTop: 3
}
}
```

9.4. Comments Widget

This widget displays open text responses from B2B customers, for example from a relationship survey.

This widget:

- Provides comments based on the date range, and filters.
- Configures the comments and additional columns.
- Allows you to export the contents to Excel.
- Allows you to view responses by question (select from a drop down).
- Allows the users to drill down to a Response View to see details of all responses.

Figure 83 Example of the Comments widget

9.4.1. Comments Code Example

Below is an example of the code to configure a Comments widget:

```
title "Comments_widget"

config hub {
hub: 432
table survey = p1027835.response
}

page "Account Page" {
widget comments { // This widget has the same properties as the List
widget. In addition it is wrapper above List widget
label: "Comments" //Label for widget
table: survey: // Table for comments
size: small //size of widget
sortColumn: comments //sorting column
sortOrder: descending // sorting order
column response comments{
footer: survey:interview_end // the information displayed at the end of
cell with comment
header: ((survey:AccountOwner + " - ") + survey:SalesRegion) // the
information displayed at the top of cell with comment
}
group question { // definition of question, a specially VPath to comment
comment: survey:Q2 // VPath for comment
label: "LTR" // the question label
}
group question { // additional columns can be added to this object. This
will result in the columns being added for only this particular question
comment: survey:Q5
label: "Technology"
column metric LTR {
```

```
view: viewLTR
format: formatterLTR
target: 10
value: average(score(survey:Q1))
}
}
column metric LTR { // definition of other columns, in this case the
column metric. You can add as many columns as necessary, and types of
this columns are not fixed
target: 10
value: average(score(survey:Q1))
}
}
}
```

9.5. Contacts Widget

The Contacts widget displays three tabs, each of which displays a list of contacts who have received survey invitation(s), and additional contact information on survey responses, respondent roles, comments etc.. The tabs are:

- **Recently responded** - lists the contacts who have responded to the survey. This tab displays the date of the last response as well as the comment. You can see the full response by clicking the View Response link. To view detailed contact information, click the View Contact link. The table can be sorted by First Name, Last Name and Last Response by clicking the relevant header.
- **All contacts** - lists all contacts who have received the invitation. The table can be sorted by First Name, Last Name by clicking the relevant header.
- **Not responded** - lists the contacts who have not responded yet. This tab displays the date when the invitation was sent. First Name, Last Name and Invitation Date by clicking the relevant header.

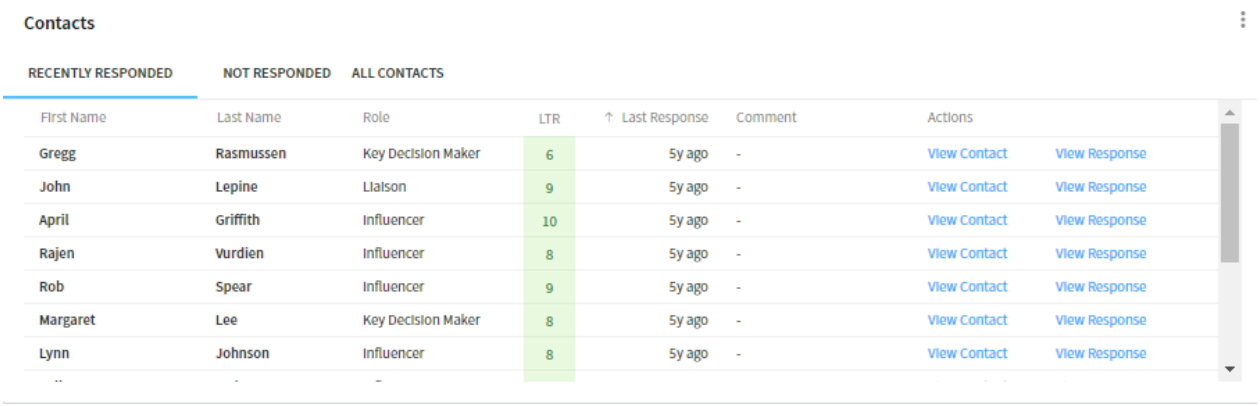


Figure 84 The Contacts widget

- **View Contact** - click this link to view the detailed information on the relevant contact in the Account tab.
- **View Response** - click this link to

9.6. KPI Widget

This widget displays Key Performance Indicator data in a gauge format. The widget provides a visual indicator for target value and support for multiple threshold and color combinations

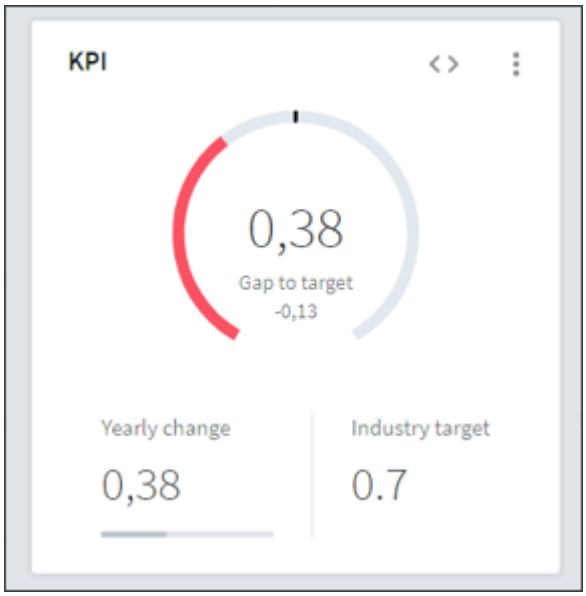


Figure 85 Example of the KPI widget

9.6.1. KPI Code Example

Below is an example of the code used to create and set up a KPI widget:

```
custom properties cp {
// filters to limit the data
currentPeriodFilter: survey:interview_start > 2016-01-01
previousPeriodFilter: survey:interview_start <= 2016-01-01
npsTarget: 9
}
widget kpi {
label: "NPS Score"
size: small
tile kpi {
label: "NPS"
value: NPS(survey:Q1) * 100
target: @cp.npsTarget
min: -100
max: 100
format: formatterLTR
targetFormat: formatterLTR
gaugeColorFormat: kpiColorFormatter
// valueColor
tile value {
label: "Responses"
value: count(survey:Q1, @cp.currentPeriodFilter)
max: count(survey:response, @cp.currentPeriodFilter)
format: integer
}
tile value {
label: "Yearly change"
value: average(score(survey:Q1), @cp.currentPeriodFilter) -
average(score(survey:Q1), @previousPeriodFilter)
//value: average(score(survey:Q1),@currentPeriodFilter) -
average(score(survey:Q1),@previousPeriodFilter)
format: formatterLTR
}
}
```

```
}
```

9.7. Markdown Widget

This widget allows you to display Rich HTML without having to write HTML, so you can have the advantages without experiencing any underlying security issues. Markdown with Github Flavor is supported - Visit https://github.github.com/gfm/ to get more details on GitHub Flavored Markdown. You can used it to enter static texts, headings, tables, images, and lists to your dashboard. This can be useful for example for providing additional information to the dashboard users.

9.8. Portfolio Breakdown by Role Widget

This widget presents an at-a-glance overview of how the Roles, for example the Decision Makers, Influencers and Users, measure up to the Key Metric for all the accounts in their Portfolio. The widget displays a breakdown by role, of detractors, promoters and passives among the respondents. It supports single, numeric, date and grid variables with some limitations. It also supports the NpsSegment emulated function.

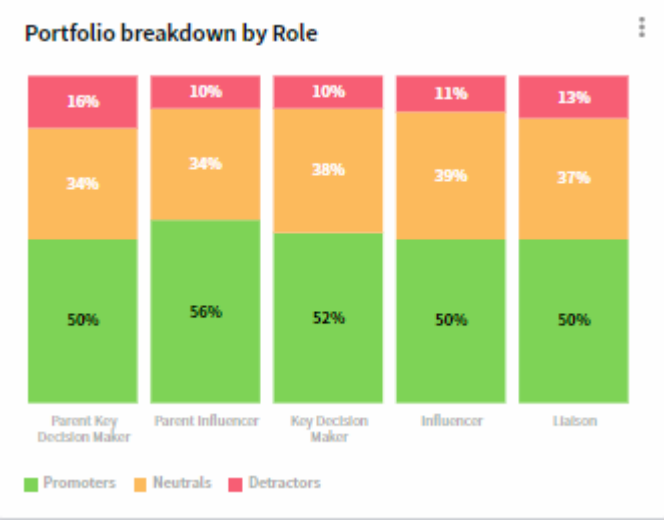


Figure 86 The Portfolio Breakdown widget

```
widget portfolioBreakdown {
label: "Portfolio Breakdown by Role"
size: small

category: contacts:ContactRole
segment: survey:NPSSegment
value: count(survey:response)
percent: on
// palette: @cp.palette
// format: floatNumber
}
```

9.9. Recent Responses Widget

This widget shows the most recent responses. This is an interesting way of bringing the recent survey activity to life for the report user.

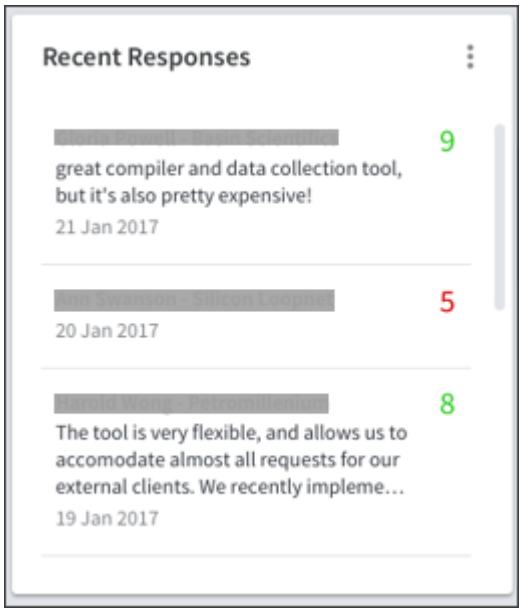


Figure 87 Example of the Recent Responses widget

9.9.1. Recent Responses Code Example

Below is an example of the code used to create and set up a Recent responses widget:

```
widget recentResponses yy {
showHeader: true
sortColumn: res
sortOrder: descending
size: medium
table: survey:
navigateTo: "Response view"
view comment fff {
lines: 4
}
view metric metrics {
valueColorFormatter: valueColor
fontSize: large
backgroundColorFormatter: transparent
}
column response res {
sortBy: footer
footer: survey:interview_end
header: survey:FirstName + " " + survey:LastName + " - " +
accounts:AccountName
comment: survey:Q2
}
column metric ltr2 {
label: "LTR"
value: average(score(survey:Q1))
format: formatterLTR
target: 10
view: metrics
}
}
```

9.10. Response Rate Widget

This widget provides users with an overview of the response statuses, for example the number and % of completes and incompletes, thereby allowing them to monitor the survey response rate. This widget can also be used for monitoring non-respondents: an end user can see for example the number or % of silent accounts, and drill down to the non-responding surveys.

This widget displays the response rate as a bar chart.

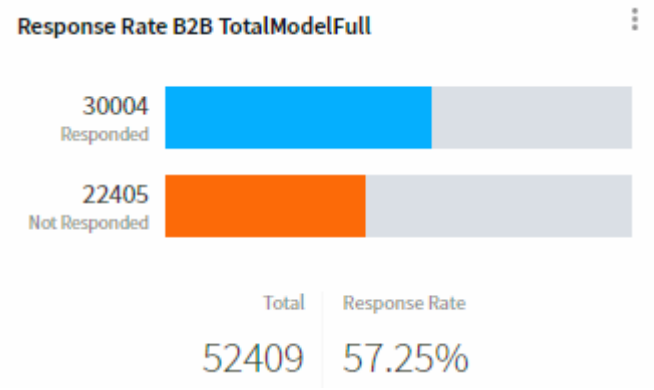


Figure 88 Example of the Response Rate widget

9.10.1. Response Rate Code Example

Below is an example of the code used to create and set up a Response Rate widget:

```
config hub {
...
// derived variable for response rate widget
variable singleChoice responseStatus {
table: survey:
label: "Status"
value: IIF(@cp.isResponded, "r", IIF(@cp.isFailedInvite, 'f', 'n'))
option code {
code: "r"
label: "Responded"
}
option code {
code: "n"
label: "Not responded"
}
option code {
code: "f"
label: "Failed invites"
}
}
}

custom properties cp {
// for response rate
isResponded: survey:status = "complete"
isFailedInvite: respondent:noOfEmailsSent != 1
}

page "Overview" {
widget responseRate {
size: small
label: "Response rate"
```

```
tile statuses {
breakBy: survey:responseStatus // derived variable in config hub block
value: count(survey:)
chart: 'bar' // 'pie'
palette: '#228e80', '#4899c4', '#c4484c'
format: valueFormat
percentFormat: formatterRR
}

tile value {
label: "Invitations"
value: count(survey:)
format: metricFormat
}
tile value {
label: "Response rate"
format: formatterRR
value: 100 * count(survey:, @cp.isResponded) / count(respondent:)
}
}
}
```

9.11. Risk Assessment Widget

This widget allows you to determine how much of a group of accounts' future revenue or another valuable measure is at risk, is safe, or is in an unknown state. The data can be grouped by renewal date, industry segments, products etc.

Combining CX and financial data, this widget offers a valuable insight to the Account Management in B2B companies. This can be helpful to get an overview across a business area to understand the monetary value of churn.

The widget displays a bar chart and a table.

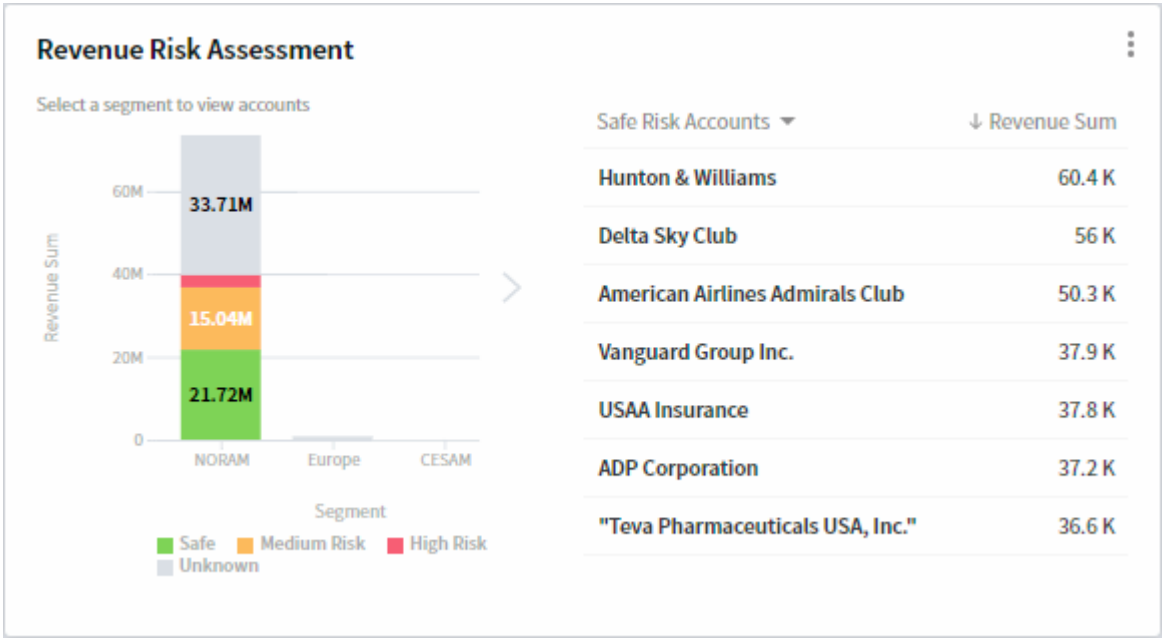


Figure 89 Example of the Risk Assessment widget

A bar chart displays how the account-related valuable measure is distributed across the risk scale in discrete risk segments. These can range from 'Safe' to 'High Risk', and are displayed in different predefined colors. The account is assigned a risk segment based on the account's Health Check value. Accounts for which no Health Check values are available are assigned the Unknown segment as their risk level cannot be defined.

In the event an additional summary breakdown measure such as renewal date is used in the widget, the bar chart may contain multiple bars according to the breakdown.

The table displays a predefined number of top accounts (by the measure under analysis), falling to the selected risk segment and breakdown measure's category. Click the relevant segment in the relevant bar to view its top accounts list in the table.

9.11.1. Risk Assessment Code Example

Below is an example of the code used to create and set up a Risk Assessment widget:

```
widget riskAssessment {
// datatable: MyTable
label: "Revenue Risk Assessment"
breakdownBy: accounts.segment //region //segment
breakdown: "1-6"
size: medium
statistic: sum
detailNav: "Account"
statistic.title: "Revenue Sum"
breakdownBy.title: "Segment"
formatters.currency.scaled: "siprefixformatter, {value} {power?}"
formatters.currency.scaledCompact: "siprefixformatter, {value}{power?}"
}
```

9.12. Search Widget

The Search widget allows searching for different entities such as contact name, accounts etc.

The user types the search string in the input field and the search is performed automatically. As the user clicks one of the displayed search results he/she is taken to a corresponding report page that displays information on the selected search result.

```
widget search {
table: contacts:
layoutArea: "header"
value: @cr.fullContactName
navigateTo: "Contact"
}
```

9.13. Summary Widget

This widget shows a compact view of aggregations of various types of data from the hub.

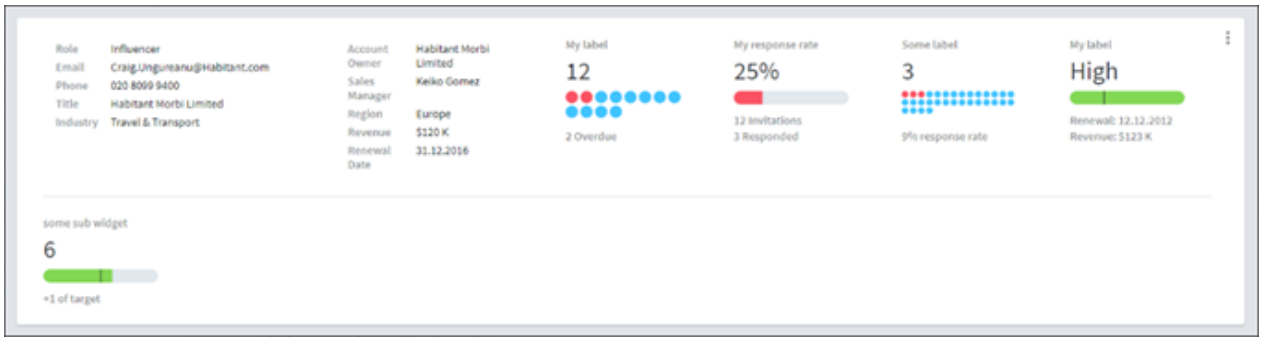


Figure 90 Example of the Summary widget in use

9.13.1. Summary Code Example

Below is an example of the code to configure a Summary widget.

```
page account "Account" {
widget summary {
size: large
table: contacts:

tile contactDetails cc {
role: contacts:ContactRole
email: contacts:email
phone: contacts:Phone
title: contacts:AccountName
industry: accounts:Industry
}

tile accountDetails cc4 {
accountOwner: contacts:AccountName
salesManager: accounts:SalesLeader1
region: accounts:WorldRegion
revenue: accounts:AnnualAccountValue
renewalDate: accounts:RenewalDate
}

tile casesStatus {
open: 12
overdue: 2
label: "My label" //not required
view: casesStatusDefaultView //not required
}

tile responseRate {
invites: 12
responses: 3
label: "My response rate" //not required
view: responseRateDefaultView //not required
}

tile surveyResponses {
total: 32
completed: 3
label: "A label" //not required
view: surveyResponseDefaultView //not required
}

tile risk {
value: 12
textValue: "High"
target: 3
renewal: "12.12.12"
revenue: "123333"
max: 10 //not required
min: 0 //not required
label: "My label" //not required
view: renewalRiskDefaultView //not required
}

tile metric a {
label: "A sub-widget" //not required
value: 6
target: 5
max: 10 //not required
min: 0 //not required
view: metricDefaultView //not required
}
}
```

```
}
```

9.14. Title Widget

The Title widget can display contact name, role and company, and supports navigation to the Account page. It can be used as a page header on the Account Overview or Contact Overview pages, or for any other page if the layout rules suit the requirements. This widget takes the characteristics of the layout area.

Figure 91 Examples of the Title widget in use

9.14.1. Title Code Example

Below is an example of the code to configure a Title widget.

```
page contact "Contact" {
mainTable: contacts:
widget title {
table: contacts:
layout column {
layout row {
layout column {
tile icon {
value: "some url for icon"
navigateTo: "sese"
}
}
layout column {
layout row {
tile value {
value: (contacts:FirstName + " ") + contacts:LastName
}
tile role {
value: contacts:ContactRole
}
tile value {
value: contacts:Title
}
}
layout row {
tile company {
value: contacts:AccountName
navigateTo: "Account1"
}
}
}
}
layout row {
tile value {
value: "Phone: "
}
tile value {
value: contacts:Phone
}
}
layout row {
```

```
tile value {
value: "Account owner: "
}
tile value {
value: accounts:AccountOwner
}
}
}
}
```

9.15. Top Accounts Widget

This widget displays the list of the most significant accounts based on a metric, for example LTR, OSAT or Health.

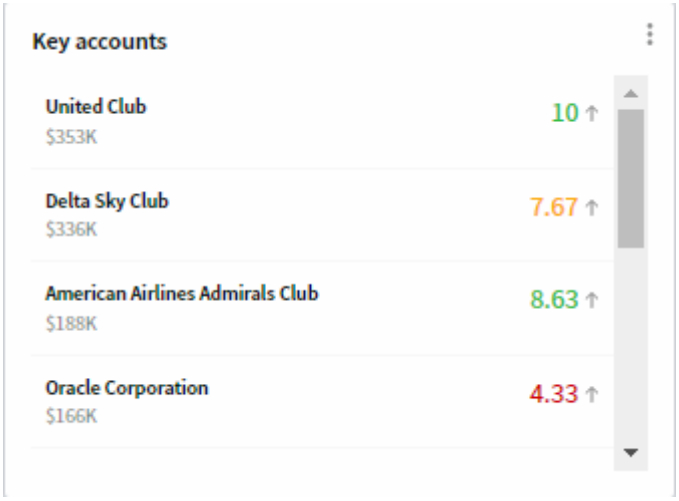


Figure 92 Example of the Top Accounts widget

9.15.1. Top Accounts Code Example

Below is an example of the code used to create and set up a Top Accounts widget:

```
widget topAccounts {
table: accounts:
size: small
sortColumn: main
navigateTo: "Account"
hierarchy: accounts:ParentAccountID
view metric metrics {
valueColorFormatter: valueColor
fontSize: medium
}
column accounts main {
accountName: accounts:AccountName
revenue: accounts:AnnualAccountValue
value: accounts:AnnualAccountValue
}
column metric ltr {
value: average(score(survey:Q1), @cp.currentPeriodFilter)
previous: average(score(survey:Q1), @cp.previousPeriodFilter)
target: @cp.ltrTarget
format: formatterLTR0
view: metrics
}
}
```

9.16. Trend Widget

This widget shows a line graph displaying how specified value series changes over time.

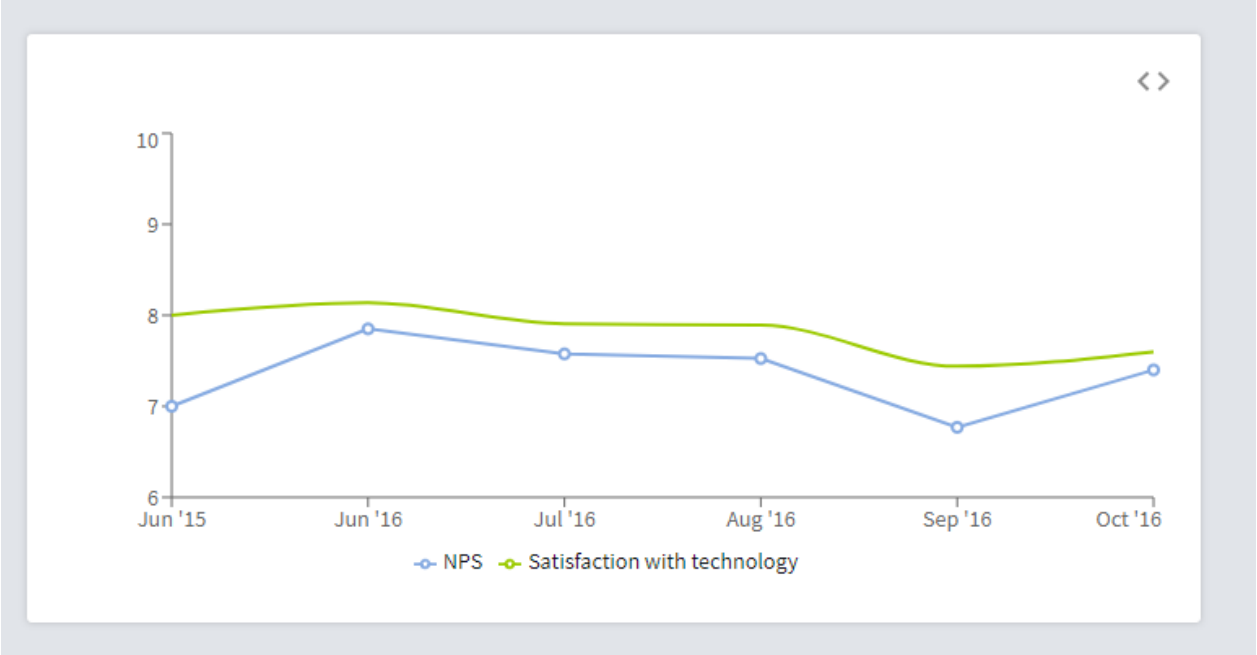


Figure 93 Example of the Trend widget

9.16.1. Trend Code Example

Below is an example of the code used to create and set up a Trend widget:

```
custom properties cp {
palette:
"#86ABE2","#F9BF00","#F18500","#EF6300","#F30000","#AA0010","#C0C0C0"
}
widget trend {
size: medium
timeline: CalendarMonth(survey:interview_start)
dateFormat: dateFormat
format:floatNumber
minValue: 0
maxValue: 10
palette: @cp.palette //
"#86ABE2","#F9BF00","#F18500","#EF6300","#F30000","#AA0010","#C0C0C0"
series value s1 {
label: "LTR, avg"
dot: true
value: average(score(survey:Q1))
}
series value s2 {
type: 'monotone'
label: "Satisfaction with technology "
value: average(score(survey:Q4))
}
}
```

10. Exporting the Report

Users and End users can export the report pages to PDF (see Exporting to PDF on page 106 for more information), and the report data from "list" type widgets to Excel (see Exporting to Excel on page 106 for more information).

10.1. Exporting to PDF

To export to .PDF, code must be added to the CDL. This code must be included in the **config report {}** block.

```
pdfExportBeta:true
```

To support and show any applied filters and expanded nodes in List widgets, include the code below in the block:

```
pdfExportBetaMode:htmlToPdf
```

```
142  config report cr {
143    pdfExportBeta: true
144    pdfExportBetaMode:htmlToPdf
145    formatter number long {
146      emptyValue: " "
147      shortForm: true
```

Figure 94 Example of the code

Then when the report is published in Live mode, the **Export to PDF** button is available in the upper-right corner of the page

Figure 95 The Export to PDF button in the live report

1. Click the **Export to PDF** button.

   A message opens across the bottom of your display, asking you what you want to do with the exported file.

2. Click **Open / Save / Save as** as required.

Note that only the current report page is exported.

10.2. Exporting to Excel

Note: Only the data from "list" type widgets can be exported to Excel; non-list widgets cannot export to Excel.

Exporting to Excel requires no special code in the DSL - all "list" widget types allow you to export the data by default.

Note that in the event the list contains a lot of data, the export will be limited to a maximum of 1000 lines. You should then use filters to show only the data you are interested in.

For a list widget:

1. Click the **Additional options** icon ⋮ to open the drop-down menu.

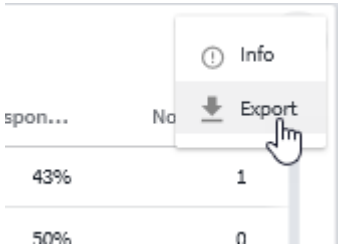
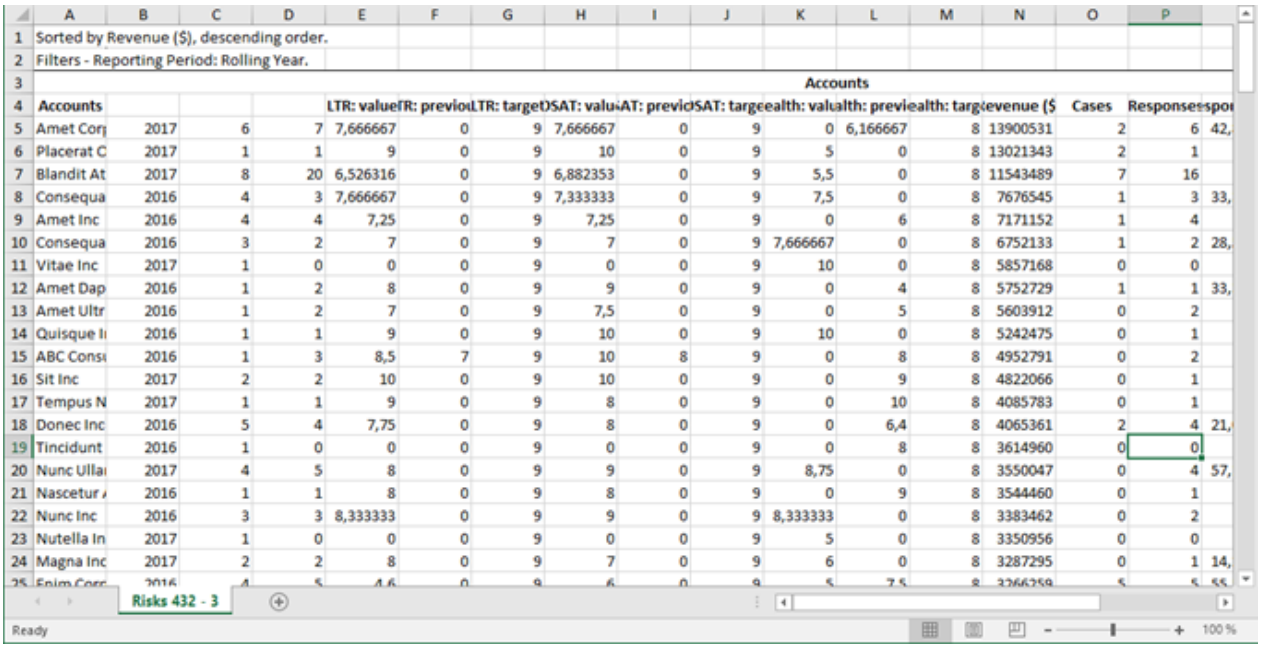


Figure 96 Exporting a list widget to Excel

2. Click **Export**.

   A message opens across the bottom of your display, asking you what you want to do with the exported file.

3. Click **Open / Save / Save as** as required.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
1	Sorted by Revenue ($), descending order.																
2	Filters - Reporting Period: Rolling Year.																
3											Accounts						
4	Accounts				LTR: value	FR: previo	LTR: target	SAT: valu	AT: previo	SAT: targe	ealth: valu	alth: previe	ealth: targ	evenue ($	Cases	Responses	spo
5	Amet Corp	2017	6	7	7,666667	0	9	7,666667	0	9	0	6,166667	8	13900531	2	6	42,
6	Placerat C	2017	1	1	9	0	9	10	0	9	5	0	8	13021343	2	1	
7	Blandit At	2017	8	20	6,526316	0	9	6,882353	0	9	5,5	0	8	11543489	7	16	
8	Consequa	2016	4	3	7,666667	0	9	7,333333	0	9	7,5	0	8	7676545	1	3	33,
9	Amet Inc	2016	4	4	7,25	0	9	7,25	0	9	0	6	8	7171152	1	4	
10	Consequa	2016	3	2	7	0	9	7	0	9	7,666667	0	8	6752133	1	2	28,
11	Vitae Inc	2017	1	0	0	0	9	0	0	9	10	0	8	5857168	0	0	
12	Amet Dap	2016	1	2	8	0	9	9	0	9	0	4	8	5752729	1	1	33,
13	Amet Ultr	2016	1	2	7	0	9	7,5	0	9	0	5	8	5603912	0	2	
14	Quisque I	2016	1	1	9	0	9	10	0	9	10	0	8	5242475	0	1	
15	ABC Cons	2016	1	3	8,5	7	9	10	8	9	0	8	8	4952791	0	2	
16	Sit Inc	2017	2	2	10	0	9	10	0	9	0	9	8	4822066	0	1	
17	Tempus N	2017	1	1	9	0	9	8	0	9	0	10	8	4085783	0	1	
18	Donec Inc	2016	5	4	7,75	0	9	8	0	9	0	6,4	8	4065361	2	4	21,
19	Tincidunt	2016	1	0	0	0	9	0	0	9	0	8	8	3614960	0	0	
20	Nunc Ulla	2017	4	5	8	0	9	9	0	9	8,75	0	8	3550047	0	4	57,
21	Nascetur A	2016	1	1	8	0	9	8	0	9	0	9	8	3544460	0	1	
22	Nunc Inc	2016	3	3	8,333333	0	9	9	0	9	8,333333	0	8	3383462	0	2	
23	Nutella In	2017	1	0	0	0	9	0	0	9	5	0	8	3350956	0	0	
24	Magna Inc	2017	2	2	8	0	9	7	0	9	6	0	8	3287295	0	1	14,
25	Enim Corp	2016	4	5	4,6	0	9	6	0	9	5	7,5	8	3266259	5	5	55

Risks 432 - 3

Ready

100 %

Figure 97 Example of a data export to Excel

11. Errors and Troubleshooting

The following sections present a number of error types that you may experience, and possible solutions.

11.1. Compiler and Runtime Errors

While you work in the Studio editor, any changes you make are saved after a few seconds. Studio checks the saved code continuously to ensure it does not contain errors, and if it notices anything unexpected it will present error messages to tell you that something is wrong. The error messages are displayed in red across the bottom of the CDL Editor panel.

Note: To reduce distraction while you are typing the code, compiler errors are suppressed in the line you are currently typing in and the new code will be assessed once you move out of that line. However the code you are currently writing may create discrepancies elsewhere in the report. For example, typing a { character into a line will mean that a corresponding } character is required at some point later in the report.

The general classes of error messages that may be presented are:

- **Compiler errors -** A compiler error occurs when the CDL syntax is incorrect, for example a required field is missing or there is a typing error in the code. The compiler for Studio reports "knows" what to expect inside each block of the code, so if it encounters something unexpected it will notify you with an error message presented at the bottom of the CDL Editor panel. Click the up-arrow in the message to open the message. Information about the error is then presented along with a very useful **Go to** link. The error message will normally contain enough information so you can understand the reason for the error, and how to rectify it.
- **Runtime errors -** when a Runtime error is presented, it usually means that something went wrong with the data request.

In each case, the first information provided is the actual location of the error, given as [row,position], in this case row 5 position 17. This is followed by examples of what should be at that location, then finally what has actually be found there, in this case the = character.

Figure 98 Example of a compiler error message

When you click **Go to line**, the CDL Editor panel moves to show the line in which the error is located. In addition, the row is highlighted and the error location is underlined.

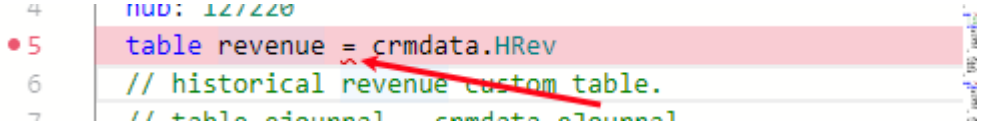


Figure 99 The line containing the perceived error is highlighted and the error location is underlined

Sometimes a compiler error will be presented that lacks complete information. For example:

```
Expected "/*", "//", "config", "import", [\n\r
;], application, end of input, identifier, title, use, or whitespace but
"}" found.
```

In this case it appears there is a problem with brackets; the compiler found a **}** character where there shouldn't be one. This could for example indicate that you are missing an opening bracket {, or you have too many } characters in total.

Tip!
Good practice (that may save your sanity) is to **indent the code**! This makes the code look neater, and it is much easier to spot if something is out of place.

11.2. Widgets Present No Data for End Users

If the end user can access your report but the widgets do not present any data, check that the end user list is linked to the same hub that is providing the data for the report. If this is not linked correctly then even though the end user can access the report, no data will be displayed in the widgets.

12. INDEX

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W